IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 21, 2021, accepted July 16, 2021, date of publication August 3, 2021, date of current version August 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3102157

Scalable and Multifaceted Search and Its Application for Binary Malware Files

DONGHOON KIM[1], JUNNYUNG HUR[2], (Graduate Student Member, IEEE), AND MYUNGKEUN YOON[2], (Member, IEEE)

[1]Kia Corporation, Seoul, Seocho-gu 06797, Republic of Korea
[2]Department of Computer Science, Kookmin University, Seoul, Seongbuk-gu 02707, Republic of Korea

Corresponding author: MyungKeun Yoon (mkyoon@kookmin.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) through the Ministry of Science and ICT (MSIT), Government of Korea, under Grant NRF-2020R1A2C1006135, and in part by the Institute for Information and Communications Technology Promotion (IITP) through MSIT, Government of Korea, under Grant 2018-0-00429 (Fundamental Research on Real-Time Similarity Measurement for Automatic Analysis of Big Data in Security) and Grant 2020-0-01826.

ABSTRACT Malicious binary files are a serious threat to industrial information systems. Because of their large number, an automatic assistant tool becomes essential for analysis, and finding similar files would be a great help. In this paper, we present a fast, scalable, and multifaceted search scheme to find similar binary malware files. We use a content-defined chunking algorithm to convert a file into a feature set for the first time. The proposed scheme uses MinHash to reduce any feature set of any file to a fixed size, which significantly improves search accuracy, processing speed, and space utilization. We theoretically prove that the new scheme returns similar files in jaccard index order. Through implementation and experiments with 12 million malicious files, we confirm that the search speed is increased by 600%, space is reduced by 90%, and the accuracy is increased by 400% at least, compared with the state-of-the-art of Elasticsearch.

INDEX TERMS Elasticsearch, inverted index, jaccard index, malware, MinHash.

I. INTRODUCTION

Cyber security is essential for stable and secure industrial information systems. According to a recent report, cybercrime would damage the world economy by approximately 6 trillion dollars annually by 2021 [1]. Malicious-software (malware) files are critical threats to cyber security, and malicious binary, or executable files have always been the most serious threat and popular attack vector that compromise servers and employee PCs in industrial enterprise networks [2]. In this paper, we focus on Microsoft Windows portable executable (PE) files for binary files because malicious PE files represent by far the prevalent security threat. A modern malware infection may start by other file types such as portable document format (PDF) or web pages, but in most of the cases the last stage of the infection is done by PE files [3]. We emphasize that the idea of this paper can be applied to any binary file including PE.

The number of malware files has been rapidly increasing to form big data. More than one million suspicious files are collected everyday by VirusTotal, the largest malware collection point in the world [2]. Because the manual analysis of malware files is so expensive, 10 hours per file [4], human and computing resources should be carefully assigned. Considering the large number of malware files, it would be better to use as many automatic assistant tools for malware analysis as possible, and there already exist such tools as static analysis, dynamic analysis, machine-learning based detection, to mention a few [1], [3], [5].

We assume that malware analysts have built a collection of malware files that have been successfully analyzed and the detailed reports exist as records, as in service providers for malware analysis [2] or antivirus companies [6]. When a suspicious sample of a PE file is given, malware analysts first check if the sample is in the history records with its file signature computed by cryptographic hash functions. If there is a match, the new analysis would be unnecessary; otherwise, the analysts continue to find those malware files from the collection that are similar to the given suspicious sample. In many cases, finding similar files that have been analyzed a prior can give valuable information concerning the analysis of the new sample, which can significantly reduce time and efforts [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Junaid Arshad.

However, finding similar malware PE files is a challenging problem. First, file representation and metrics for file similarity can be defined differently. If a PE file is represented by its byte sequence as a long string, two PE files can be compared in terms of a string comparison metric. If a PE file is represented by a set of its internal human-readable strings, two PE files can be compared in terms of a set comparison metric. Therefore, two PE files can be similar on the basis of a byte sequence, but different on the basis of ASCII string sets. Second, the similarity of two files is compared in pairs, which means that all collected files should be compared with a given suspicious sample. This is how the state-of-the-art of ssdeep runs to find similar files [7], which is not scalable and requires much computing resources.

In this paper, we present a new automatic assistant tool for malware analysis that can search a large collection of malicious PE files for the most similar file when a suspicious PE sample is given. We call this MUltifaceted Search Engine Using MinHash sampling (MUSEUM) that can find similar files in terms of different criteria against a query file in the order of jaccard index. We implement MUSEUM in the Elasticsearch platform [8], which successfully indexes more than 10 million files in commodity PCs. We use the MinHash sampling algorithm [9] for Elasticsearch indexing and search that not only reduces index size but also retrieves similar files in the order of jaccard index, which has not been achieved by any previous work. The main contributions of MUSEUM are summarized as follows:

- For a given query file, MUSEUM returns a list of similar files in jaccard index order. General binary files as well as human-readable documents can be indexed and searched in MUSEUM.
- A file is converted into a set of keywords, and an inverted indexing is established from the set to the file. We apply the MinHash algorithm to extract only a fixed number of keywords (e.g., 128) for indexing. Therefore, the index space decreases and the search speed is increased significantly, which also solves the scalability problem.
- In MUSEUM, similarity between two files can be measured by various criteria; for example, an executable file can be converted into a set of byte *n*-grams or a set of its imported libraries. We call each criterion as a feature, and the set of keywords as the feature set. An executable file can be converted into a byte stream *n*-gram feature set, a string feature set, an imported function feature set, etc.
- We implemented MUSEUM with two commodity PCs where more than 10 million malicious PE files were successfully indexed. Experimental results demonstrate that search speed is increased by 600%, space is reduced by 90%, and accuracy is increased 400% as measured by relative errors, compared with the state-of-the-art inverted indexing of Elasticsearch [8] and ssdeep indexing [7], [10].

We explain the limitations of MUSEUM as well. In this paper, we present a new kind of automatic assistant tool that can find previous similar files fast and accurately in terms of different features. We emphasize that this new tool cannot solve all the challenging problems with malware analysis, such as obfuscation, packing, fileless malware, sneaky connection to command and control servers, etc. For example, the proposed scheme can find only similar files at byte-sequence level when byte *n*-grams are used as a feature. This definitely cannot find semantically similar files in terms of malicious behavior, which is usually the final goal of malware analysts. However, MUSEUM can save their time and efforts by automatically and mechanically looking up a large collection of previously analyzed files.

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

II. PROBLEM AND MOTIVATION

Because finding similar files is essential for critical applications, researchers have presented fuzzy hash algorithms to solve the problem. For example, ssdeep is one of the most widely used fuzzy hash algorithms; the whole byte sequence of a file is divided into non-overlapping sub-strings, and each sub-string is independently hashed [7]. Some bits from each hashed value are extracted and concatenated to make a representative string for the file. In Fig. 1 (a), file1 is divided

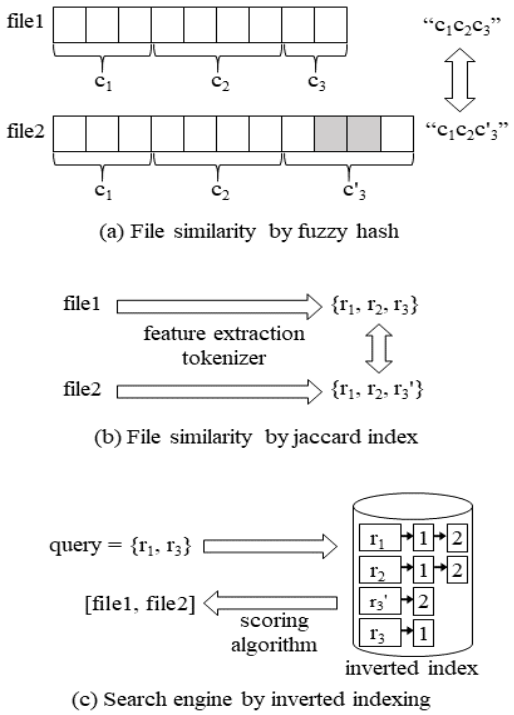

FIGURE 1. Comparison of file similarity measurement by fuzzy hash (a) vs jaccard index (b). Representative strings are compared in fuzzy hash while a jaccard index is computed between two feature sets. For a limited number of query keywords, inverted indexing can retrieve relevant files (c).

into three parts and "$c_1c_2c_3$" becomes the representative string for file1. To measure the similarity between file1 and file2, their representative strings are compared to produce their similarity score. There are some limitations with fuzzy hash. First, as the similarity should be measured in pair-wise, fuzzy hash cannot search for similar files in a large scale. Second, files can be compared only in terms of byte sequence. In this paper, we want to measure the file similarity in jaccard index, but ssdeep does not work for a set.

A PE file can be represented with different features. A feature is a basis property to represent a PE file, such as a byte sequence, a set of human-readable strings embedded in the file, a set of API calls, etc. For example, PE files include a set of imported libraries that the operating system provides. It is known that comparing the sets of imported libraries can give decisive hints to malware analysis. If two sets are almost the same and include unusual imported libraries in common, we may conclude that they are closely related to each other, or the same attacker may have invented them. Once we convert a file into a feature set, we can compute the jaccard index between them, which ranges from 0 to 1. Note that the jaccard index between two sets is defined as the ratio of the size of their intersection set to the size of their union set. However, this approach also requires pair-wise comparisons against every file in the dataset. Fig. 1 (b) shows that two files are converted into their feature sets whose jaccard index is 0.5(=2/4).

Inverted indexing is a cornerstone for document search in a large dataset [11], and Elasticsearch is a de facto open platform standard [8]. Inverted indexing provides fast search for relevant documents against a set of query keywords as shown in Fig. 1 (c). In this case, file1 and file2 are retrieved, and file1 is recommended as the most relevant document because file1 includes both r_1 and r_3. A search engine has a unique scoring algorithm to give each retrieved document a different recommendation score. Elasticsearch uses the BM25 scoring algorithm as a default setting. Inverted indexing skips the time-consuming pair-wise comparisons against all dataset files, which makes MUSEUM a fast and scalable solution. However, we cannot use a naive inverted indexing algorithm for MUSEUM; first, naive inverted indexing does not retrieve similar files in the order of jaccard index. We need a new scoring algorithm to solve the problem. Second, inverted indexing assumes that the set size of query keywords should be moderate for search performance. For example, the default maximum size of query keywords is 1,024 in Elasticsearch [12]. However, a feature set of a file can include more keywords than this size. For example, the size of the byte 4-gram feature set from 1 MB file can be as large as 999,997 (= 1,000,000 – 3).

III. MUSEUM: MULTIFACETED SEARCH ENGINE USING MinHash SAMPLING

We introduce the overall architecture of MUSEUM first. Then, each component of the converter and MinHash-based tokenizer is explained in detail.

A. OVERVIEW

MUSEUM is a search engine for finding similar files against a query file. One or more criteria for measuring two files in jaccard index are defined first. We explain MUSEUM in indexing and searching phases. During the indexing phase, all files are converted into feature sets for each criterion. Only a fixed number of elements from a feature set are selected by the MinHash-based tokenizer. Then, an inverted indexing is built for each measurement criterion. During the searching phase, a file is given as a query file. The same converter and tokenizer are applied to this file, which results in a fixed number of query keywords. Using the keywords, the search engine returns a list of similar files in jaccard index order. Fig. 2 shows this multifaceted indexing of MUSEUM. Note that an indexing is established for each similarity measurement criterion, or feature, independently. We emphasize that the similarity of malware files can be compared in terms of different features.

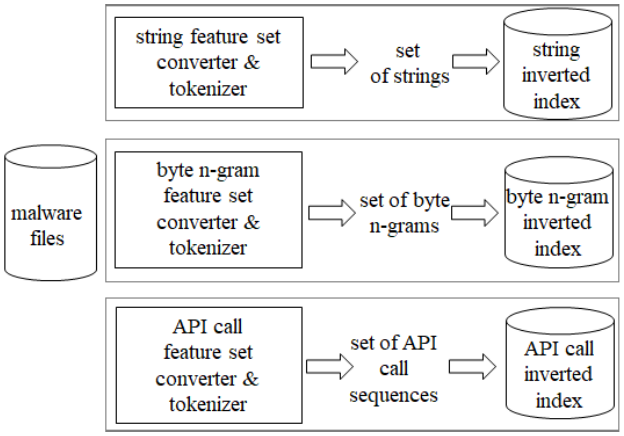

FIGURE 2. MUSEUM for multifaceted indexing and search that can find similar malware files in terms of different criteria, or features.

B. FEATURE SET CONVERTER

An executable malware file can be converted into different feature sets. The simplest conversion would be a set of byte n-grams from the file; the byte sequence of the file can be divided into multiple non-overlapping subsequences of size n in bytes. For more complicated feature sets, if we use a static analysis tool [13], a set of ASCII strings, or a set of imported libraries can be extracted from a file [3]. If we use a dynamic analysis tool [14], API call sequences can be generated from a file. Each of them can be a criterion when file similarity is measured. In this paper, we determine to use the jaccard index as a measurement tool. Therefore, we need to convert any sequence-type feature into a feature set. For example, an API call sequence is a long string that should be converted into a set.

We adopt a content-defined chunking algorithm for the first time to convert a sequence-type feature into a set for the purpose of indexing and search of malicious PE files [15]. We choose an asymmetric extremum (AE)

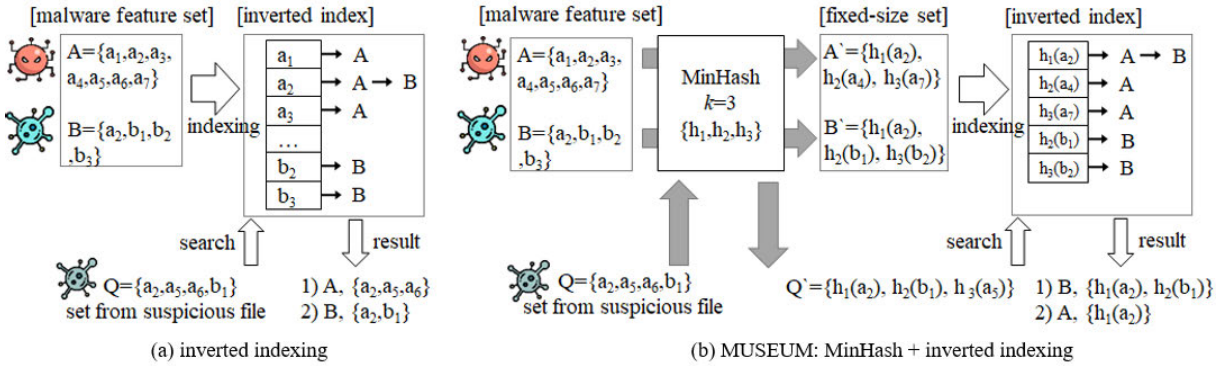

FIGURE 3. Comparison of (a) inverted indexing and (b) MUSEUM. The size of any feature set is reduced to a fixed number k with the MinHash-based tokenizer in MUSEUM.

chunking algorithm because this runs quickly and divides any sequence into evenly-sized chunks around the average chunk size [16]. After an average chunk size is fixed as c, a byte sequence of length m will be converted into a set of no more than m/c distinct elements with a high probability. Note that naive n-grams would generate a worst-case set of $(m-c+1)$ elements. From 1 MB file, 1 million (exactly $2^{20}-3$) 4-gram words would be generated in the worst case. Therefore, AE chunking efficiently converts any sequence into a feature set, which is confirmed in our experiments in the next section. The byte sequence of a file can also be converted into a feature set through this chunking algorithm.

C. MinHash-BASED TOKENIZER

We first explain how the MUSEUM tokenizer converts feature sets of any size into fixed-size ones through MinHash sampling. Then, we prove that this can estimate the jaccard index between a query file and each indexed file. Assume that a malware feature is already selected for similarity comparison and feature sets have been extracted from malware files. For example, 10,000 malware files are provided for indexing, and we select an ASCII string feature for similarity measurement. This means that we have 10,000 ASCII string sets for the malware files.

In this paper, we use A and B to represent the feature sets of two malware files that are indexed; A' and B' are their fixed-sized versions from A and B after MinHash is applied. We call these dashed as representative sets. Similarly, Q is the feature set of a query file, and Q' is its representative set. A, B, and Q are of variable size while the representative sets are of the same fixed size. We explain indexing phase first and then search phase.

1) INDEXING

The tokenizer uses k-independent hash functions of $h_1, \ldots, h_k$ to select only k elements from a variable-sized feature set. For a given feature set, the smallest value of $h_i(a)$ is selected for all a's that are included in the feature set. Because there are k hash functions, we collect the minimum $h_i(a)$ for h_i to form a representative feature set of size k. Then, all k elements of the representative set are used for indexing. In MUSEUM, we implement inverted indexing on the Elasticsearch platform. Therefore, every malware file is indexed with its representative k hashed values for a certain feature criterion. The relationship between original feature set A and its representative set A' becomes

$$A' = \bigcup_i \{min\,\{h_i(a)|a \in A\}\} \tag{1}$$

where $|A'|$ is always k unless no hash collision occurs. Fig. 3 compares the difference between the classical inverted indexing and MUSEUM; feature sets A and B are obtained from two malware files that have 7 and 4 feature elements, respectively. The inverted indexing uses all elements for indexing, whereas MUSEUM selects a fixed number of elements first and then builds indexing. In Fig. 3, 10 elements of $\{a_1, \ldots, a_7, b_1, \ldots, b_3\}$ are used for the classical inverted indexing, but only 5 elements of $\{h_1(a_2), h_2(a_4), h_3(a_7), h_2(b_1), h_3(b_2)\}$ for MUSEUM.

Consequently, MUSEUM stores a small number of hashed words for indexing that saves significant storage space. For example, the average number of ASCII strings from a malware file is 2,417 for our experimental dataset, but 128 hash functions ($k = 128$) are sufficient for accurate jaccard index estimation in MUSEUM, which is later explained in detail.

2) SEARCH

When a query file is given, we convert it into a representative set of size k similar to the indexing process described previously. These k elements are considered keywords for the document search in Elasticsearch that returns a list of malware file IDs in order of its scoring algorithm results. The default scoring algorithm of Elasticsearch is Okapi BM25, shortened to BM25, where BM is an abbreviation of best matching [8]. BM25 is based on the probabilistic retrieval framework and a term frequency–inverse document frequency (tf–idf) style document retrieval algorithm [11], [17]. For a given query $Q = \{q_1, q_2, \ldots, q_n\}$, the feature set of a query file, the BM25 score of document A, and the

feature set of a malware file is defined by:

$$s(A, Q) = \sum_{i=1}^{n} IDF(q_i) \times \frac{f(q_i, A) \times (\alpha + 1)}{f(q_i, A) + \alpha \times (1 - \beta + \beta \times |A|/avgDL)} \quad (2)$$

where $f(q_i, A)$ is the term frequency in A for each q_i, $|A|$ is the length of A, and $avgDL$ is the average document length in the document collection [17] that is the collection of feature sets in this paper. Although we can configure α and β as free parameters in equation (2), BM25 cannot return similar malware files in jaccard index order.

In MUSEUM, the purpose of search is to find similar malware files in jaccard index order. To solve this problem, we define a new scoring algorithm as follows:

$$s(A', Q') = |A' \cap Q'|. \quad (3)$$

Equation (3) only measures the intersection size of A' and Q', which can be easily implemented in the Elasticsearch platform. Because every malware feature set is converted into a fixed-size set through MinHash, we only consider such A' that includes any $min\{h_i(q)|q \in Q\}$. We always need to consider only k query words because $|Q'| = k$ for any Q.

We prove that the scoring equation (3) preserves the jaccard index order. For this, we use the Broder's theorem of $Prob(min\{h_i(q)|q \in Q'\} \in A' \cap Q') = j(A, Q)$ [9]. Since we use k independent hash functions, there are always k queries in Q'. For simplicity, we ignore hash collisions, and we can use adequate hash functions to make the collision probability negligibly small. Then, the measured size of $|A' \cap Q'|$ or Equation (3), becomes an estimator for $k \times j(A, Q)$. Therefore, $|A' \cap Q'| > |B' \cap Q'|$ implies $k \times j(A, Q) > k \times j(B, Q)$, and $j(A, Q) > j(B, Q)$.

Therefore, in MUSEUM, converting a malware feature set into a MinHash-based fixed-size representative set and measuring the intersection set size guarantees finding similar files in jaccard index order.

We explain the example of Fig. 3 where measuring $|Q' \cap A'|$ can find the most similar malware file based on the jaccard index, but measuring $|Q \cap A|$ cannot preserve this property. Fig. 3 (a) depicts how simple inverted indexing finds the most similar file. Two malware files are preprocessed to produce their feature sets, A and B. For simplicity, we use all the elements from the feature sets to build an inverted indexing table. When a query file is given, its feature set is generated in the same way, denoted as Q. The scoring algorithm here is the intersection size between Q and each of indexed files, and therefore A is chosen as the most similar malware file. Note that all of the four feature elements of Q are used and only their postings are checked. On the contrary, a MinHash tokenizer is adopted by MUSEUM as shown in Fig. 3 (b). All feature sets, including Q, are converted into representative feature sets of the same size k. We build the inverted indexing table with only k elements from each of the representative feature sets. Note that only k hashed values from Q' make a set of query keywords, and scoring equation (3) finds B as the most similar malware file in the right order of jaccard index.

D. JACCARD INDEX ESTIMATOR

We develop an estimator for the jaccard index in MUSEUM. We assume that all malware feature sets are converted into a fixed size. We define an indicator random variable of $I\{min\{h_i(q)|q \in Q'\}, A'\}$ for indexed document A', query file Q', and hash function h_i; this variable becomes 1 if A' is included in the posting list of $min\{h_i(q)|q \in Q'\}$ or otherwise becomes 0. Subsequently, equation (4) becomes an estimator for the jaccard index between A and Q according to Broder's theorem [9], which is denoted as $\hat{J}(A, Q)$:

$$\hat{J}(A, Q) = \frac{\sum_{i=1}^{k} I\{min\{h_i(q)|q \in Q'\}, A'\}}{k} \approx \frac{|A \cap Q|}{|A \cup Q|} = j(A, Q) \quad (4)$$

The equation takes the average of the jaccard index estimations k times; Broder found that $j(A, Q) = P(min\{h(a)|a \in A\} = min\{h(q)|q \in Q\})$. We demonstrate that a moderate size of k, e.g. $k = 128$, results in accurate jaccard index estimation based on the experiments described in Section IV.

IV. EXPERIMENTS

We evaluate MUSEUM through extensive experiments using datasets of up to 12 million malicious PE files. We implement MUSEUM on the Elasticsearch platform with two PCs. The experimental goal is to confirm that MUSEUM can find similar malware files in jaccard index order in terms of various features. We compare MUSEUM with two rival schemes: (1) the default inverted indexing of Elasticsearch [8] and (2) ssdeep with n-gram-based inverted indexing, shortly ssdeep indexing [7], [10]. We emphasize that few methods can index and search a large number of malware files. Although these rival schemes were not designed for measuring metrics of jaccard index, they can index and search a large volume of malware files on the Elasticsearch platform. Therefore, we compare MUSEUM with them.

The first rival scheme, inverted indexing, is to use the default inverted indexing setting provided by Elasticsearch. The difference between this and MUSEUM is that MinHash sampling is implemented for MUSEUM as a tokenizer. The simple inverted indexing cannot find similar malware files in jaccard index order. Moreover, greater storage space is required to hold all the elements of a feature set as keywords for indexing.

The second rival scheme, ssdeep indexing, is adopted by some security industries as a practical search technique to find similar malware files in terms of sequences [7], [10]. The main idea is to use ssdeep, n-gram, and Elasticsearch together. However, this scheme supports only sequence-type features, such as file byte streams and API call sequences. Two representative strings are generated for every malware file through ssdeep, de facto fuzzy hashing, and a set of n-grams generated from two representative strings. Then, inverted indexing is

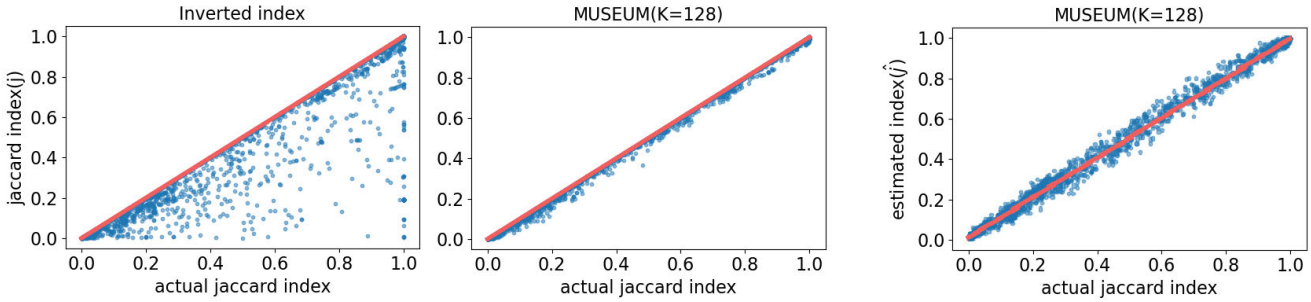

FIGURE 4. Accuracy comparison of inverted indexing and MUSEUM where dataset1 and the ASCII string feature are used. Each point represents a query file, whose x-coordinate is the true jaccard index between the query file and the most similar malware file; the y-coordinate of the left and middle plots is the jaccard index between the query file and the first ranked search result while the right plot uses the estimator of equation (4). If the search is perfect, all points would be on $y = x$.

built on Elasticsearch. More explanations are available in [7]. This practical scheme is scalable because a malware file is converted into a small set of keywords. However, our experimental results show that MUSEUM outperforms this rival scheme in accuracy.

A. EXPERIMENTAL SETUP

We implemented MUSEUM on Elasticsearch with two high-end PCs with Intel i9-7920 processors and 128 GB of memory. One master node and three data nodes are configured for the Elasticsearch deployment [8].

The dataset is collected from a commercial malware site from 2018-10-01 to 2019-02-01 [18]. For experiments, we use two different collecting periods for indexing and query; dataset1 consists of one day of indexing files from 2018-10-01 and one day of query files from 2018-10-02. Dataset2 consists of 124-day indexing, from 2018-10-01 to 2019-02-01, and one day of query files from 2019-02-02. The numbers of indexed malicious PE files of dataset1 and dataset2 are 68,858 and 12,000,000, respectively. The numbers of query malware files of dataset1 and dataset2 are 3,000 and 10,000, respectively.

We use dataset1 to compare MUSEUM, inverted indexing, and ssdeep indexing, for two different feature types of ASCII strings and byte streams. The byte stream feature can be obtained by either n-grams or AE chunking. The moderate size of dataset1 enables complete pairwise comparison for a given query file, and therefore we can know the ground-truth of the best search results. This enables us to evaluate the correctness of MUSEUM and its rival schemes. After confirming the soundness of MUSEUM search with dataset1, we use dataset2 for evaluating the big data processing capability of MUSEUM. Pairwise comparison is not possible for dataset2 because of its large size. Therefore, we do not know the ground truth for accuracy, but we can confirm the scalable processing capability and usefulness of search results from MUSEUM. Table 1 summarizes these two datasets and features.

We use three comparison metrics of, accuracy, throughput, and storage space. Accuracy is measured as the exactness of search results. For a given query file, each search scheme including MUSEUM will return an ordered list of search results. We use only the first ranked malware file. Then, we can compute the jaccard index between the query file and the search result file. If we know the actual most similar file for the query file, we can compute the relative error of the jaccard index. Throughput is measured as the number of query files processed per second. Space is measured as the storage size to keep indexing information.

TABLE 1. Summary of two malware datasets and features.

Type		dataset1	dataset2
Features		Byte stream (n-gram), Byte stream (AE), ASCII string	Byte stream (AE)
Indexing	Period	2018.10.01	2018.10.01 ∼ 2019.02.01
	no.of files	68,858	12,000,000
Query	Period	2018.10.02	2019.02.02
	no.of files	3,000	10,000

B. MUSEUM VS INVERTED INDEXING

The first set of experiments compare MUSEUM with the default inverted indexing of Elasticsearch. We use dataset1 for the experiments and a set of ASCII strings extracted from a malware file is the feature. We use an open-source string extractor to obtain ASCII strings [19]. To find ground truth information, we exhaustively compute the jaccard index between every query file and all indexed files; the total number of pairwise comparisons is 206,574,000 (= 68,858 × 3,000), which is not possible for the dataset2 that is much larger than dataset1.

The number of hash functions, k, is set to 128 for MUSEUM, and the smallest hashed value for each hash function is selected for the representative feature set. We observe that $k = 128$ is a good parameter setting for MUSEUM among a range of k values. For inverted indexing, we use all ASCII strings for the feature set without sampling.

The search accuracy is compared in Fig. 4 where inverted index and two MUSEUM versions are presented from left to right. For the left and middle plots, each point represents

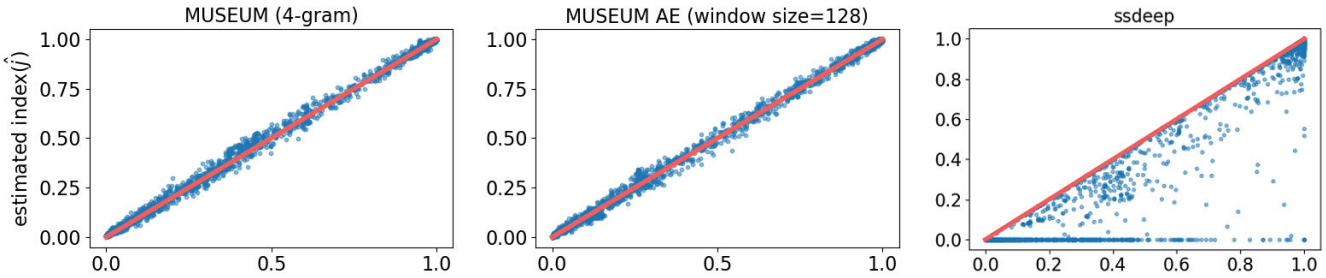

FIGURE 5. Accuracy comparison of MUSEUM with byte stream (4-gram) feature (left), MUSEUM with byte stream (AE chunking) feature (middle), and ssdeep indexing (right) [6]. The dataset1 is used for the experiments.

a query file where x-coordinate is the true jaccard index between the query file and the most similar file, and y-coordinate is the jaccard index between the query and the first ranked malware file that is returned as the search result of the query file. The right plot presents another MUSEUM version where the estimator of equation (4) is used, which quickly provides the estimated result. If the search result is perfect, the point would be on the $y = x$ line. As in Fig. 4, MUSEUM has higher accuracy than inverted indexing. The relative errors of the two schemes are 0.120 and 0.030, respectively.

To compare time and space costs, we measure the throughput and storage space of the inverted index and MUSEUM. For indexing, MUSEUM is faster than inverted indexing because it uses only 128 elements for the representative feature set. We confirm that the throughput of MUSEUM increases by more than 600%, the space is reduced by more than 90%.

C. MUSEUM VS SSDEEP INDEXING

In this set of experiments, we compare ssdeep indexing and MUSEUM using the dataset1. Because ssdeep indexing can handle only a sequence-type feature, we use the byte stream feature. Fig. 5 depicts the accuracy plots for MUSEUM with 4-gram (left), MUSEUM with AE chunking (middle), and ssdeep indexing (right). Two MUSEUM versions outperform ssdeep indexing.

In MUSEUM, a sequence-type feature should be converted into a set, and then the set is converted into a representative set of size k through implementing MinHash. In this paper, we adopt the AE chunking [16], a content-defined chunking method, to convert a sequence into a set for the first time in the search and inverted indexing literature.

For the byte stream feature, we apply the AE algorithm to divide each byte stream of a malware file into several non-overlapping byte sub-streams of different lengths (chunks); the feature set is a collection of these chunks. The chunking algorithm uses a sliding window to compute content-defined delimiters to divide the variable-length chunks. The repeated chunks of the same content can be treated as one element in the feature set. Therefore, this chunking algorithm outperforms the classical n-gram scheme in reducing set size.

As in Fig. 5, we confirm the accuracy plots of MUSEUM (4-gram) and MUSEUM AE look similar. However, the throughput of AE is much higher than that of n-gram by 7,000%. We set n to 4 because 4-gram performs most effectively among n-grams. The average number of n-grams is 357,638, and the number of AEs is 2,731.

The average window size of MUSEUM AE is a tuning parameter for the AE chunking. A larger window size implies that the size of chunks increases, but the number of chunks decreases. Therefore, a large window size increases throughput, but this might decrease accuracy. However, in the next experiments, we confirm that a large window size does not significantly lower the accuracy. Because AE chunking outperforms n-gram, we use AE chunking for MUSEUM in the remaining experiments.

D. MUSEUM PARAMETERS

The number of hash functions, k, and the average chunk size of AE chunking, c, are important parameters for MUSEUM. They may affect the accuracy and throughput of MUSEUM. Therefore, we show experimental results with different k and c, which are shown in Figs. 7 and 8, respectively. According to the AE chunking algorithm [16], w is the tuning parameter of c as $c = (e - 1) \times w$ where e is the natural constant. We use w instead of c in the figures. The dataset1 and the feature of AE chunking are used for this experiment.

The experimental results show that k affects both accuracy and throughput as in Fig. 7. We use $k = 128$ as a default value. On the other hand, the search accuracy does not degrade much when w increases. This is because the size of malware files for our experiments is generally much larger than w. For example, if a file size is 1 MB and $w = 256$, we still have $10^6/((e - 1) \times w) \approx 2,284$ chunks. This number is large enough for MinHash to keep its high accuracy. In this paper, we set $w = 128$ as a default value for AE chunking.

E. EXPERIMENTS ON LARGE DATASETS

In the final experiments, we use dataset2, which consists of more than 12 million malware files collected for 124 days. The feature is a byte stream, with AE used as

a content-defined chunking algorithm. We use MUSEUM to build the indexing. Simple inverted indexing without implementing MinHash fails to even build indexing because of the large number of files and the large size of feature sets. The query files are randomly selected from one day of data from 2019-02-02 to create 10,000 queries. For comparison of throughput and space of different-sized datasets, we used datasets of 60,000, 1,000,000, 6,000,000, and 1,2000,000 malware files, respectively. The first and the last are dataset1 and dataset2, respectively. The second and the third are subsets of dataset2. The experimental results are plotted in Fig. 6.

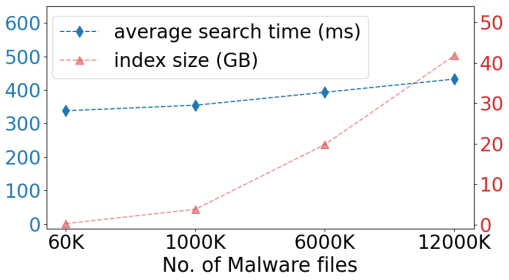

FIGURE 6. Comparison of throughput and indexing space with different collection sizes of malware files. Dataset1, 2 and byte stream (AE chunking) feature are used.

The throughput of search and the space for indexing are compared in Fig. 6, where four different indexing sizes are presented. We confirm that the throughput can be preserved steadily even with 12 million malware files indexed. In contrast, indexing space increases with the number of files. This figure shows that MUSEUM indexing is very efficient; the indexing space is less than 70 gigabytes, whereas the original malware files amount to 9.34 terabytes. This scalability is enabled by the use of MinHash sampling and the Elasticsearch open-source platform.

V. RELATED WORK

Broder studied the problem of the resemblance among files and coined MinHash in his seminal study [9]. MinHash can estimate the jaccard index among files with a fixed number of selected elements from the original sets. His work affects the research on finding similar files and documents, which is especially useful for web search [20], [21]. MinHash solves the problem of a large number of elements in a set, but the scalability problem caused by a large number of files persists. Locality sensitive hash can solve this problem by first observing hash collisions of file subparts and then selecting candidates for similar files [22]. In this paper, we combine MinHash and Elasticsearch to find similar malware files in jaccard index order in a scalable way. In this paper, we use only the first-ranked search result, but our schemes can be easily extended to use several of the high-ranking results. This is similar to the traditional k-nearest neighbor problem [11]. Although recent work solves the problem efficiently [23], [24], none of the studies can find similar files in jaccard index order, to the best of our knowledge.

Measuring jaccard index is an effective way to compare malware files. Raff and Nicholas used Minhash and compression algorithms to estimate the similarities among malware files in terms of byte sequences [25], [26]. The main idea is to compress two concatenated files. If the compression rate is high, the two files can be considered very similar; otherwise, they are not similar. However, this algorithm still requires every pairwise comparison among files, which can be solved by MUSEUM.

Finding similar malware files is a critical task for modern malware analysis. The number of similar malware files steadily increases because of malware variants. After systematically dissecting a large number of malware datasets collected over a long period, Ugarte-Pedrero *et al.* find that few malware samples are unique [3]. Interestingly, not only malware files but also benign files have similar cousins [4]. Therefore, finding similar files, not only text files but also executable files, becomes important for cybersecurity. This can present malware analysts with decisive materials for analysis and an advanced starting point instead of analysis from scratch, significantly reducing their time and effort.

Machine learning and recent deep learning can detect variant malware files. After collecting a large dataset of malware files and their stable labels, or malware family names, machine learning models are trained to predict if a suspicious file is malware or benign or which malware family a suspicious file belongs to; in general, supervised learning is widely used [27], [28]. Cui *et al.* adopt deep learning models from the image-processing literature to solve the problem of malware classification [29]. Raff *et al.* present an end-to-end deep learning model that does not need any preprocessing or additional feature engineering for malware files [30]. Antivirus companies understand the limitations of traditional malware detection technologies. They actively study and adopt machine learning algorithms to detect variant malware files [6].

Finding malware variants has been a critical research topic in cybersecurity. Shafiq *et al.* introduce how to use data mining to detect variant malware files by using header information from Windows PE files [13]. Diro and Chilamkurti present a cyber-attack detection method based on deep learning models, deployed at edge nodes in fog computing [31]. However, all of these machine learning and deep learning-based detection systems cannot find similar malware files, especially in jaccard index order.

Malware analysts know that similar malware files of a high jaccard index provide decisive hints for malware analysis, but no scalable solutions have been developed until now. Some industry experts use ssdeep, a de facto fuzzy hash function, together with Elasticsearch to develop practical industrial technologies [7], [8]. This is a clever idea because similar files can be found by ssdeep, and scalable and fast search can be performed by the industry-verified Elasticsearch, which removes pairwise comparison operations. However,

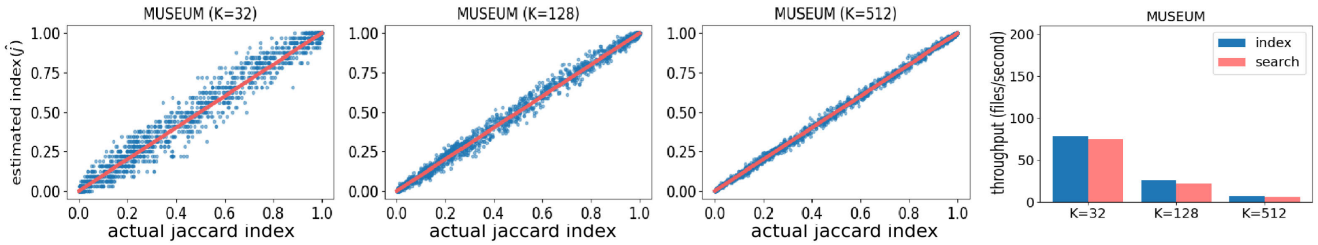

FIGURE 7. Different numbers of hash functions and their effects on accuracy and throughput for MUSEUM. The dataset1 and byte stream (AE chunking) feature are used for the experiments.

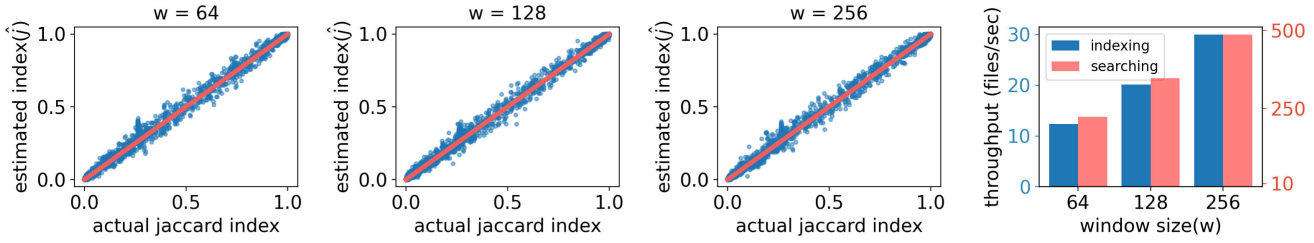

FIGURE 8. Different average chunk sizes of AE chunking and their effects on accuracy and throughput for MUSEUM. The dataset1 and byte stream (AE chunking) feature are used for the experiments.

the soundness of this technique is not theoretically proven, and the technique does not guarantee that similar files can be returned in jaccard index order.

Fuzzy hash algorithms can efficiently find similar files [10], [32]. They divide a file into multiple non-overlapping parts and each part is encoded into a small string or bitmap. For similar two files, some parts of the string or bitmap would be identical. However, fuzzy hash algorithms require inefficient pairwise comparisons of all the small strings or bitmaps. This problem can be relaxed with inverted indexing [7].

VI. CONCLUSION

In this paper, we presented a new search engine that can find similar malicious binary files in jaccard index order for a query file. Its unprecedented accuracy and scalability derive from a new method to combine MinHash and inverted indexing. The similarity can be defined differently based on multiple features, and our proposed scheme can reflect the multiple features individually or collectively. We proved the accuracy of our proposed scheme theoretically and also confirmed, through extensive experiments on real malware datasets up to 10 million files, that it outperforms previous work and current state-of-the-art methods.

REFERENCES

[1] O. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[2] (2021). *Virus Total*. Accessed: Jan. 22, 2021. [Online]. Available: https://www.virustotal.com/en/statistics/

[3] 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.

[4] B. Li, K. Roundy, C. Gates, and Y. Vorobeychik, "Large-scale identification of malicious singleton files," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2017, pp. 227–238.

[5] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Comput. Surveys*, vol. 52, no. 6, pp. 1–28, Jan. 2020.

[6] *Machine Learning for Malware Detection*, Kaspersky Lab, Moscow, Russia, 2019.

[7] Intezer. (2021). *SSDEEP-Elastic*. Accessed: Jan. 22, 2021. [Online]. Available: https://github.com/intezer/ssdeep-elastic

[8] (2021). *Elasticsearch*. Accessed: Jan. 22, 2021. [Online]. Available: https://www.elastic.co

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

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

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

[12] (2021). Search settings. Accessed: Jan. 22, 2021. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/7.5/search-settings.html#indices-query-bool-max-clause-count

[13] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: Mining structural information to detect malicious executables in realtime," in *Proc. Recent Adv. Intrusion Detection*, 2009, pp. 121–141.

[14] (2021). *Cuckoo Sandbox 2.0.7*. Accessed: Jan. 22, 2021. [Online]. Available: https://cuckoosandbox.org/

[15] 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.

[16] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, and Y. Zhou, "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1337–1345.

[17] (2021). *Okapi BM25*. Accessed: Jan. 22, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Okapi_BM25

[18] (2021). *Virussign*. Accessed: Jan. 22, 2021. [Online]. Available: https://www.virussign.com/

[19] GNU. (2021). *GNU Strings (GNU Binutils for UBUNTU) 2.30*. Accessed: Jan. 22, 2021. [Online]. Available: https://sourceware.org/binutils/docs/binutils/strings.html

[20] E. Cohen and H. Kaplan, "Summarizing data using bottom-k sketches," in *Proc. 26th Annu. ACM Symp. Princ. Distrib. Comput. (PODC)*, 2007, pp. 225—-234.
[21] M. Bury, C. Schwiegelshohn, and M. Sorella, "Similarity search for dynamic data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 11, pp. 2241–2253, Nov. 2020.
[22] J. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets*, 2nd ed. New York, NY, USA: Cambridge Univ. Press, 2014.
[23] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proc. VLDB Endowment*, vol. 12, no. 5, pp. 461–474, Jan. 2019.
[24] M. Aumüller, E. Bernhardsson, and A. Faithfull, "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Inf. Syst.*, vol. 87, Jan. 2020, Art. no. 101374.
[25] E. Raff and C. Nicholas, "Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash," *Digit. Invest.*, vol. 24, pp. 34–49, Mar. 2018.
[26] E. Raff and C. Nicholas, "An alternative to NCD for large sequences, Lempel-Ziv Jaccard distance," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1007–1015.
[27] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. USENIX Secur.*, 2019, pp. 729–746.
[28] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," 2018, *arXiv:1804.04637*. [Online]. Available: http://arxiv.org/abs/1804.04637
[29] Z. Cui, X. Fei, X. Cai, C. Yang, G. G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3187–3196, Jul. 2018.
[30] M. Al-Fawa'reh, A. Saif, M. T. Jafar, and A. Elhassan, "Malware detection by eating a whole APK," in *Proc. 15th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Dec. 2020, pp. 1–7.
[31] A. Abeshu and N. Chilamkurti, "Deep learning: The frontier for distributed attack detection in fog-to-things computing," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 169–175, Feb. 2018.
[32] F. Breitinger and I. Baggili, "File detection on network traffic using approximate matching," *Digit. Forensics, Secur. Law*, vol. 9, no. 2, pp. 23–36, 2014.

DONGHOON KIM received the B.S. and M.S. degrees in computer engineering from Kookmin University, South Korea, in 2018 and 2020, respectively. He works with Hyundai Kia Motors as a Data Scientist. His research interests include information security and big data analysis.

JUNNYUNG HUR (Graduate Student Member, IEEE) 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.

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 has been a Professor with the Department of Computer Engineering, Kookmin University, Seoul, since 2010. He worked with Korea Financial Telecommunications and Clearings Institute, Seoul, from 1998 to 2010. His research interests include computer and network security, security intelligence, data science for cyber security, and artificial intelligence for security.