I-Filter: Identical Structured Control Flow String Filter for Accelerated Malware Variant Classification

Taegyu Kim, Woomin Hwang, †Ki-Woong Park and Kyu Ho Park
Korea Advanced Institute of Science and Technology †Daejeon University
{tgkim, wmhwang} @core.kaist.ac.kr, and kpark@kaist.ac.kr †woongbak@dju.kr

***Abstract*—As the number of malware variants has grown rapidly, classification speed has become crucial in security issues. While several techniques for malware variant classification have been proposed, they involve a speed-accuracy trade-off. In an attempt to achieve a speedy and accurate malware variant classification, we thoroughly analyze previously proposed methods and identify a critical performance bottleneck in string-to-string matching. This paper presents and evaluates a technique called I-Filter that enhances the performance of the previous approach, approximate matching. I-Filter has the following novel mechanism, the hash-based equivalent procedure matching technique. Our performance evaluation confirms that a performance improvement of on average 1,043 times through I-Filtering.**

Keywords-malware variant classification; identical structured control flow; database;

I. INTRODUCTION

According to statistics written of the AV-TEST [6], around 80 million malwares were created in 2013. The appearance of new malwares has increased at an exploding pace. In addition, 88% of them are malware variants that are only slightly modified existing malwares [5].

To accurately classify the numerous malware variants, we need a speedy and accurate malware variant classification method. Many graph-matching based methods have been proposed. Of various methods, exact matching and approximate matching, which were proposed in Malware [5], are robust in malware variant generation techniques such as self-mutating or instruction reordering. However, exact matching shows higher performance but a lower classification rate while approximate matching is more accurate but slower. For accurate and speedy malware variant classification, we accelerated approximate matching, which is more accurate. In approximate matching, comparisons of strings through the edit distance algorithm create the main bottleneck because strings are compared character by character. In our approach, we first match identical structured control flow graphs through I-Filter using a hash-based equivalent procedure matching technique and then match the remaining graphs. This is effective since we found that malware variants have a high share ratio of identical structured control flow graphs. As a result, we gained a performance improvement of on average 1,043 times compared to matching without I-Filtering because we could match graphs largely through hash-based equivalent procedure matching instead of edit distance matching processes which is the main performance bottleneck of previous work.

We organize the remaining parts of our paper as follows: In Section II, we explain the background knowledge. In Section III, we review and analyze related work. In Section IV, we describe our approach to solve the performance bottleneck. In Section V, we evaluate the performance, accuracy and share ratio of identical control flow graphs. In Section VI, we present our conclusion.

II. BACKGROUND

In this part, we explain the background knowledge as follows: preprocessing to generate Structured Control Flow Strings (SCFS), the edit distance, definition of similarities and the previous approach.

A. Preprocessing: Decompilation and String Conversion

Before analyzing a suspicious binary, which is a pre-unpacked malware or normal program, we preprocess two step procedures to generate SCFSs representing structured control flow graphs [3]. A structured control flow graph is a type of a graph consisting of subgraphs to express control structures in a high-level language. This graph is represented by one SCFS. In the first step, we convert an input binary into high-level codes through a decompiler, the Reverse Engineering Compiler [4]. Then, we use a structuring technique to convert decompiled codes into SCFSs. During structuring, hash values for strings are generated. We describe these procedures in Figure 1 and the grammars for converting structured control flow graphs into character tokens in Table I.

978-1-4799-6444-4/14/$31.00 ©2014 IEEE

B. Edit Distance

Edit distance is a way to measure how many times are necessary to modify, insert or delete for two strings to be identical [9]. To match similar strings, edit distance is used in previous studies as well as our own. The formula for string *x* and *y* is described in Equation 1.

$$ed_{i,j}(x,y) = \begin{cases} ed_{i,0}(x,y) = i, & for\ 0 \le i \le m, \quad (1a) \\ ed_{0,j}(x,y) = j, & for\ 0 \le j \le n, \quad (1b) \\ min \begin{cases} ed_{i+1,j}(x,y) + 1, \\ ed_{i,j+1}(x,y) + 1, \\ ed_{i+1,j+1}(x,y)_{(a_i = b_j)}, \\ ed_{i+1,j+1}(x,y) + 1_{(a_i \neq b_j)}, \end{cases} & (1c) \end{cases}$$

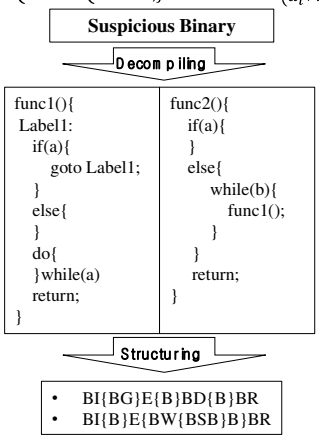

Figure 1. Preprocessing : Decompilation and Structuring

In base cases, we use Equation 1a and 1b. However, we normally apply Equation 1c. Time complexity is *O(mn)*. Both *m* and *n* represent the lengths of strings.

C. Definitions of Similarities

We present definition of similarities used in malware variant classification. We describe three types of similarities: String-to-string (S2S) similarity for two SCFSs, malware-to-malware similarity reflecting S2S similarities, and set similarity which is the final result of malware similarity.

String-to-string Similarity To compare malware similarities, we measure S2S similarities first. The similarity measurement formula [7] is described in Equation 2.

$$w_{ed} = 1 - \frac{ed(x,y)}{\max\big(len(x), len(y)\big)} \quad (2)$$

In Equation 2, *x* and *y* represent string *x* and *y*; *len(x)* is the length of string *x*; *max(len(x), len(y))* indicates that a string with a larger length is selected; and *ed(x,y)* is the result of the edit distance between string *x* and *y*. By normalizing the edit distance values with *max(len(x)*, *len(y))*, we can determine the degree of difference between the two strings. Thus, we can find w_{ed} which is S2S similarity between string *x* and *y* through the difference ratio subtracted from 1.0.

Malware-to-Malware Similarity This similarity is calculated by summating S2S similarities by weights proportional to lengths of strings. We describe its formula proposed in previous work [3] in Equation 3.

$$S_x = \sum_i \begin{cases} 0, & w_{ed_i} < t \\ w_{ed_i} weight_{x_i}, & w_{ed_i} > t, \end{cases} \quad (3)$$

x denotes one input program or malware in the malware database. We measure both an input binary and malwares in the database. x_i denotes one of string in *x*. $weight_x$ is calculated by a length of string x_i divided by the total length of a malware *x*.

Set Similarity After measuring malware-to-malware similarities of both an input binary *i* and a selected malware *d*, we multiply both similarities. We describe this formula suggested in a previous work [3] in Equation 4.

$$S(i,d) = S_i S_d, \quad (4)$$

TABLE I. GRAMMARS FOR STRUCTURED CONTROL FLOW [3].

Signature	Result	Signature	Result
Basic Block	B	Procedure Call	C
Branch 'if'	I	Branch 'else'	E
Return	R	Break	X
Continue	C	Goto	G
PreTested Loop	W	PostTested Loop	D
Infinite Loop	F	Branch Condition '&&'	A
Branch Condition '‖'	Z	Open Brace '{'	O
Close Brace '}'	P		

D. Malware variant classification in Previous Work

A previous study [3] used preprocessing. Then, its system measures similarities through matching SCFSs with pre-analyzed strings in malware databases. Based on the immediate similarity, it selects a candidate family similar to an input binary. If a candidate family is selected, it matches the SCFSs of an input only with malwares in the candidate family. Then, it decides whether an input is malicious or not based on the similarity result. Figure 2a describes these steps with a flow chart, and Figure 2b shows the matching states in each step.

III. RELATED WORK

To classify malware variants, many studies have proposed graph-matching based malware variant classification. Filtering methods for string matching acceleration have also been proposed for performance improvement. In this part, we describe existing graph-matching-based malware variant classification and string filtering for performance improvement to reveal our contributions to this endeavor.

A. Graph-based malware variant classification

Malware variant classification using SCFSs has been proposed [3][5]. According to their approach, they utilize structuring to approximately match procedure-level graphs. Then, this system measures similarities to existing malwares in malware databases to find the most similar malware. Based on their experiments, it is capable of classifying malware variants considering performance or accuracy. Two matching methods are suggested: a fast method called exact matching and an accurate method called approximate matching. However, exact matching has a lower accuracy,

and approximate matching has a lower performance. In order to develop both accurate and speedy malware variant classification, we choose to accelerate approximate matching whose accuracy is higher than the other method.

B. String Filtering

Performance improvement in string matching is an ongoing issue, especially, in bioinformatics. In order to find identical proteins or nucleotides, bioinformatics engineers convert them into string sequences and match identical protein or nucleotide string sequences in huge databases. For performance improvement, they have proposed short filtering [8]. If a string shares a certain number of substrings of another string, this pair is considered identical without an actual sequence alignment because proteins or nucleotides

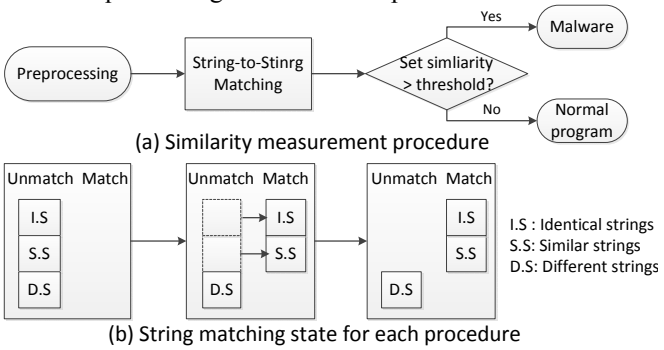

Figure 2. Flow Chart of Previous Approach.

have their own composition patterns. Even though exact prediction of a specific protein or nucleotide string based on substring matching can be used for protein or nucleotide string sequences, it cannot be used for matching SCFSs of malwares. Because patterns of character tokens in SCFSs are based on the malware author's coding style, these patterns are unpredictable only with parts of strings. However, our experiments revealed that SCFSs of malwares in the same family share identical SCFSs based on our experiments. Therefore, we utilized this characteristic to design the I-Filter to accelerate malware variant classification.

IV. OUR APPROACH

Our main goal was to accelerate approximate matching methods. We developed I-Filtering to accelerate matching procedures and changed the query policies with malware databases fit for our approach. After preprocessing, we matched identical SCFSs through hash-based equivalent procedure matching in databases. Then, we selected a candidate family if an immediate similarity exceeded a threshold value. After a candidate was selected, we matched strings through both the I-Filter and the edit distance algorithm [9] for the remaining strings. Figure 3a shows the malware variant classification procedures after the preprocessing step, and Figure 3b describes the matching states in each step. In this section, we first explain the performance bottleneck in previous studies. We then propose the I-Filter as a solution to the performance bottleneck and describe procedures of similarity measurements in the following subsections.

A. Performance Bottleneck in Previous Work

To calculate a set similarity, S2S similarities should be measured first. In this step, we consider SCFSs whose similarities of strings are higher than the threshold value of similarity. Previously, only the edit distance algorithm was used for S2S similarity measurements, but its time complexity was too large. In addition, as the lengths of strings increased, its processing time became much longer.

B. I-Filter for String-to-string Matching

In our approach, we use the I-Filter first for performance improvement and then the edit distance algorithm for measuring similarities. S2S matching refers to string pairs

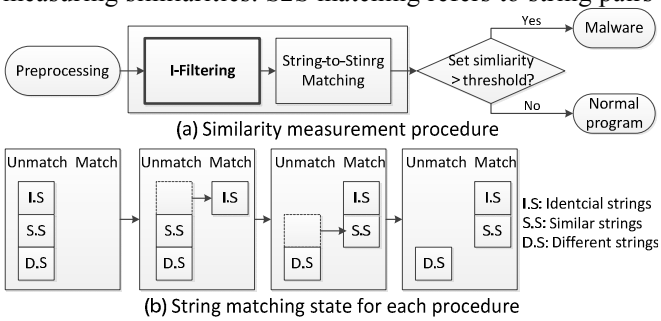

Figure 3. Overview of Malware Analysis Steps with I-Filter.

whose similarity is larger than the threshold value. For S2S matching, we first match identical SCFSs through I-Filtering using a hash-based equivalent procedure matching through hash value comparisons of SCFSs. We use CRC-64 for hash values. However, I-Filtering is limited to matching only identical SCFSs. Thus, we use I-Filtering for many matching cases and match the remaining SCFSs through the edit distance algorithm. There are two reasons why I-Filter is effective in performance enhancement: low time complexity and a high level of sharing identical SCFSs.

Lower Time Complexity One reason why I-Filtering is effective in performance improvement is that the time complexity of I-Filtering is lower than that of the edit distance algorithm. The time complexity of the edit distance between two SCFSs is $O(mn)$. Both m and n are lengths of SCFSs, and their minimum value is 10 [3]. Considering the number of SCFSs, s in the malware database, time complexity for one S2S matching is $O(mn)O(s) = O(smn)$. On the other hand, the time complexity of I-Filtering is $O(m)$. S2S matching is processed through a hash-based equivalent procedure matching whose time complexity is $O(1)$. This should be repeated until the identical SCFS is found. Because we stored the hash values in B-tree [1], the time complexity for finding one SCFS is $O(logs)$ from $O(1)O(logs) = O(logs)$. In addition, checks for identicalness are required to prevent hash collisions for S2S matching. The time complexity for hash collision checking is $O(m)$ which is proportional to the length of a shorter SCFS in a string pair. Therefore, we can induce $O(m)$ from $O(m + logs)$.

High Level of Sharing Identical SCFSs According to our experiments, many SCFSs are identical in the same family. If SCFSs are identical among malware variants, we can match identical SCFSs through I-Filter whose matching time cost is much lower than the edit distance algorithm. Thus, the matching processing time is reduced more as the number of identical strings is larger. In the case of previously analyzed malwares, all SCFSs are identical. Hence, the matching processing is done only through I-Filtering. In the case of a previously unseen malware variant, they still share many identical SCFSs. For example, *Klez*, *Netsky* and *Roron* share an average of 78, 70, 84 percent of identical SCFSs. Therefore, we can accelerate S2S matching through reduced dependence on the edit distance algorithm. Figure 4 shows S2S matching processes without I-Filtering, and Figure 5 shows the reduced number of matching using the edit distance algorithm due to I-Filter.

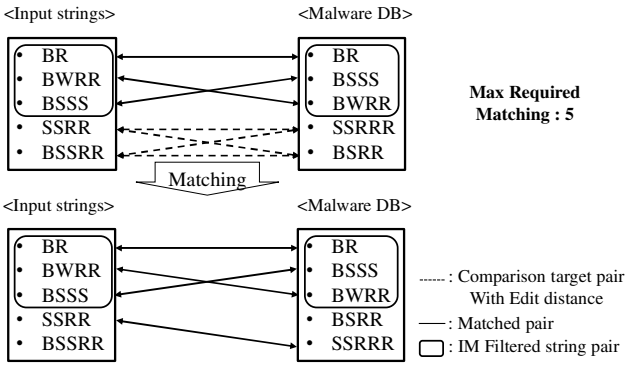

Figure 4. Previous matching. (S2S similarity threshold = 0.8)

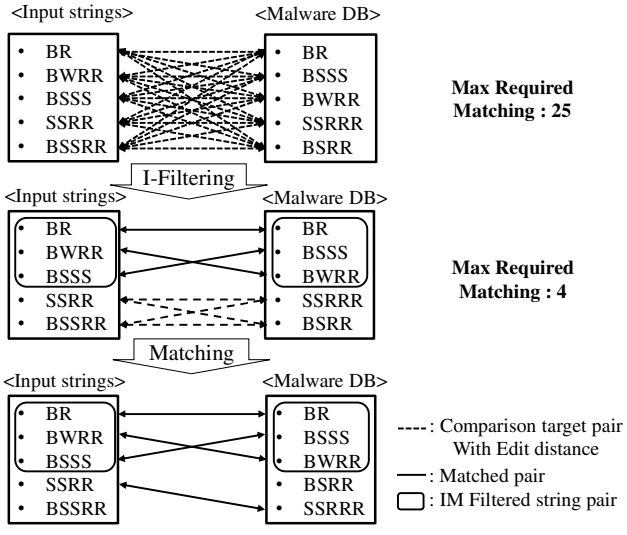

Figure 5. Matching with I-Filter. (S2S similarity threshold = 0.8)

C. Similarity Measurement with Malware Database

In this section, we describe malware databases, our matching algorithms with databases and the parameters we used. We divide the algorithm into two stages: similarity measurements with a global database and a local database. We describe both algorithms in Algorithm 1 and 2 and give detailed explanations in the following subsections.

Malware Database There are two types of databases: a global database and a local database. We used a global database to match identical SCFSs. It stores SCFSs and metadata of all malware families such as malware families, variant names and hash values. Local database stores same data but only in one specific malware family. We store indexed hash values in both types of databases since we use I-Filtering operations in both databases.

Similarity Measurements with a Global Database In the similarity measurement procedure, matching with a global database is first because which local database is used is not determined yet. With a global database, we match only identical SCFSs using I-Filtering. In this step, near unique strings are processed first, and then duplicated SCFSs are matched. Because they are not shared among many malwares, they are useful to determine a specific candidate malware. Otherwise, there is a possibility of determining a wrong malware or variant as a candidate. During similarity measurements, if the set similarity exceeds a set similarity threshold t, matching processes are performed on a local database. If it does not exceed t even after all SCFSs are matched and the highest set similarity is lower than min_t, an input binary is considered a normal program. Otherwise, the top five candidate malwares with similarities higher than the others are selected. The standard of selecting a local database depends on what is the malware candidate because a local database stores only malwares of a specific family.

Similarity Measurements with a Local Database With a local database, we apply I-Filtering first with a determined malware candidate because similarity of all SCFSs are not measured with I-Filtering. Then, similarity of the remaining SCFSs are measured through the edit distance algorithm. In order to be considered similar, the S2S similarity between two strings should exceed T. Because identical SCFSs are matched and the number of SCFSs in a local database is lower than in a global database, the number of SCFSs that should be measured with the edit distance algorithm is reduced dramatically. Therefore, the performance of our approach in similarity measurements is better than conventional approximate matching.

Parameters There are several variables that we determine. We choose min_t as 0.1 and determine the number of candidates as 5 based on our experiments. We use 0.9 for T and 0.6 for t as used in the previous work [3]. However, we can change these values according to additional experiments.

V. EVALUATION

This section presents performance improvements, similarities between variants and the share ratio of identical SCFSs. In the performance evaluation, we focused on matching performance. The experimental environment consisted of Intel Xeon quad Core 2.13GHz, 2G RAM, 32GB SSD, Cent OS 6.3 64bit and MySQL 14.14 for the database. We used 3,000 malware samples [10] and create 7,000 pseudo malware samples for the performance evaluation.

A. Performance Improvements

In this section, we evaluate the performance of approximate matching for two cases: I-Filtering and no filtering. Figure 6a and 6b show the processing time. We sorted the processing time according to malware families since the processing time is roughly proportional to the number of procedures, and malwares in the same family tend to have a similar number of procedures and structured control flows. By using I-Filter, we gained varying performance improvements from 100 times to more than 2,000 times. Even if the number of samples in the databases increased, its performance was still reasonable for fast classification. The average performance improvement was 1,043 times. As the number of malware samples in database increased, performance improvements were larger. We gained these improvements from the hash-based equivalent technique, indexed identical SCFSs on a B-tree and a high level of sharing of identical SCFSs.

Algorithm 1: Similarity measurement

t	: Set similarity threshold
min_t	: Minimum of *t*
str_i	: String of an input
str_d	: String in database
gdb	: Global malware database
ldb(malware)	: Database for malware family
Matching	: Matching & Update similarity Refer to algorithm 2
mname	: Malware name
setSim[mname]	: Set similarity with malware

Require : Initialize all variables
1: **for** *str_d in gdb(unique)* **do**
2: **if** *any setSim[mname]* ≥ t **then**
3: candidate = mname
4: Matching(str_i, str_d, setSim[mname], identical)
5: **for** *str in gdb(duplicate)* **do**
6: **if** *any setSim[malNname]* ≥ t **then**
7: candidate = malName
8: Matching(str_i, str_d, setSim[mname], identical)
9: **if** *any setSim[malwareName]* ≤ min_t **then**
10: return Result(Safe, 0)
11: **else**
12: Top5candidates = CandidSelect(setSim[mname])
13: **for** *candidate in Top5candidates* **do**
14: **for** *str in ldb(candidate, identical)* **do**
15: Matching(str_i, str_d, setSim[candidate], identical)
16: **for** *str in ldb(candidate, similar)* **do**
17: Matching(str_i, str_d, setSim[candidate] , similar)
18: **if** *setSim[mname]* ≥ t **then**
19: SelMalware = candidate
20: return Result(SelMalware, setSim[SelMalware])
21: return Result(Safe, 0)

Algorithm 2: Matching

T	: String-to-string similarity threshold
min_t	: Malware name
S_i[mname]	: Malware-to-malware similarity for an input
S_d[mname]	: Malware-to-malware similarity for a malware in DB
hash	: Hash value of a structured control flow string

Input : str_i, str_d, setSim[mname], matching
Output : Updated similarity

1: **if** *matching equal to identical* **then**
2: **if** *str_i.hash equal to str_d.hash* **then**
3: S_i += str.length
4: S_d += str.length
5: setSim[mname] = $S_i \times S_d$
6: **else**
7: **if** *stringSimilarity* ≥ *T* **then**
8: setSim[mname].S_i += str_i.length × strSimilarity
9: setSim[mname].S_d += str_d.length × strSimilarity

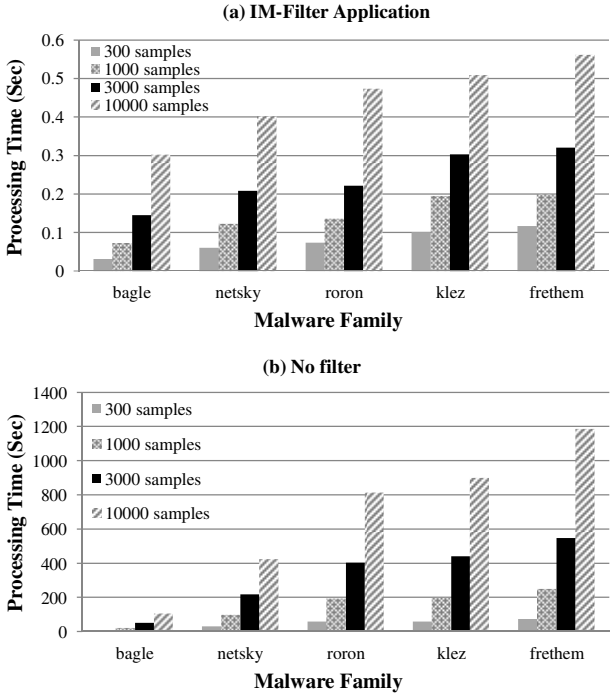

Figure 6. Processing time with I-Filtering and no filtering.

B. Similarity in the Same Malware Family

We measured similarities with malware samples available in offensive Computing [10] after unpacking them. We used the same similarity measurement method as Malwise [5] but a different decompiler [4]. Therefore, similarity results were slightly different from the related study. We also matched the same strings first and then did the same thing for similar strings. To determine whether the malwares were classified, we used the same set similarity threshold value, 0.6, used in the related work [5]. We found similarities between malware variants in *Klez*, *Roron* and

Netsky malware families. Table II shows the results. Based on experiments, *Klez*, *Roron* and *Netsky* had 43, 62, 66 percent matching rates. As the matching rates become higher, new malware variants will probably be classified with a small number of malware variants in databases. However, these matching rates do not directly indicate classification rates. For example, the matching rates of the *Klez* family were only 43 percent. However, let us suppose *a*, *b*, *c* and *d* *Klez* variants are group *A* and the other ones, *e*, *g* and *i* *Klez* variants, are group *B*. In this case, one malware samples from group *A* and the other one from group *B* are enough to classify all *Klez* malware variants in Table II. Still, it is more probable to classify new malwares with higher matching rates.

C. Share Ratio of Structured Control Flow Strings

In this section, we describe how many SCFSs are identical in the same families. As share ratios of identical strings are higher, more S2S matchings are processed through I-Filtering instead of the edit distance algorithm.

TABLE II. SIMILARITY BETWEEN MALWARE VARIANTS.

	12	25	35	37	a0	b39	b50
12		0.5	0.53	0.53	0.66	0.5	0.39
25	0.5		0.84	0.89	0.56	0.93	0.63
35	0.53	0.84		0.94	0.64	0.9	0.63
37	0.53	0.89	0.94		0.6	0.95	0.63
a0	0.66	0.56	0.64	0.6		0.57	0.43
b39	0.5	0.93	0.9	0.95	0.57		0.63
b50	0.39	0.63	0.63	0.63	0.43	0.63	

Roron

	a	b	c	d	e	g	h	i
a		0.73	0.91	0.65	0.5	0.49	0.5	0.45
b	0.73		0.8	0.87	0.54	0.53	0.54	0.52
c	0.91	0.8		0.7	0.5	0.49	0.5	0.45
d	0.65	0.87	0.7		0.52	0.5	0.52	0.51
e	0.5	0.54	0.5	0.52		0.94	0.91	0.91
g	0.49	0.54	0.49	0.5	0.94		0.93	0.92
h	0.5	0.54	0.5	0.52	0.91	0.93		0.99
i	0.45	0.52	0.45	0.51	0.91	0.92	0.99	

Klez

	ab	b	c	k	p	u	w	x
ab		0.74	0.84	0.91	0.64	0.75	0.7	0.6
b	0.74		0.76	0.72	0.54	0.58	0.55	0.53
c	0.84	0.76		0.86	0.6	0.67	0.63	0.59
k	0.91	0.72	0.86		0.61	0.7	0.66	0.58
p	0.64	0.54	0.6	0.61		0.68	0.6	0.88
u	0.75	0.58	0.67	0.7	0.68		0.85	0.64
w	0.7	0.55	0.63	0.66	0.6	0.85		0.57
x	0.6	0.53	0.59	0.58	0.88	0.64	0.57	

Netsky

Table III shows how many identical strings are shared in *Roron*, *Klez* and *Netsky*. Each value indicates the number of SCFSs of malware variants in each column is shared with malware variant in each row.

From the experiments, *Klez*, *Roron*, *Netsky* share an average of 78, 70, 84 percent of identical SCFSs. Thanks to the high level of sharing of identical SCFSs between malware variants, we can measure most S2S similarities faster using I-Filtering.

VI. CONCLUSION

Our goal was to accelerate approximate matching whose performance is too low to classify numerous malware variants. To solve this low performance problem, we have proposed I-Filter to accelerate classification. In our approach, we first match identical SCFSs through I-Filtering and remaining strings through edit distance algorithm whose computing cost is high. Due to the lower cost of I-Filtering and the high level of sharing of identical strings, we gained performance improvement of on average 1,043 times for malware variant classification.

ACKNOWLEDGMENT

The work was supported by Ministry of Knowledge Economy, Republic of Korea, (Project No. 10035231).

TABLE III. SHARE RATIO OF IDENTICAL STRUCTURED CONTROL FLOW STRINGS BETWEEN MALWARE VARIANTS

	12	25	35	37	a0	b39	b50
12		0.46	0.47	0.51	0.78	0.47	0.36
25	0.51		0.93	0.95	0.53	0.97	0.7
35	0.51	0.92		0.96	0.56	0.94	0.7
37	0.51	0.94	0.97		0.56	0.98	0.7
a0	0.82	0.52	0.54	0.54		0.52	0.4
b39	0.51	0.97	0.94	0.98	0.53		0.7
b50	0.51	0.9	0.9	0.9	0.53	0.9	

Roron

	a	b	c	d	e	g	h	I
a		0.8	0.9	0.72	0.61	0.61	0.61	0.61
b	0.72		0.86	0.85	0.67	0.66	0.67	0.67
c	0.9	0.84		0.72	0.61	0.6	0.61	0.61
d	0.81	0.92	0.8		0.69	0.69	0.69	0.69
e	0.74	0.78	0.73	0.74		0.97	0.91	0.93
g	0.74	0.77	0.72	0.74	0.97		0.93	0.95
h	0.74	0.78	0.73	0.74	0.91	0.93		0.98
i	0.74	0.78	0.73	0.74	0.94	0.95	0.98	

Klez

	ab	b	c	k	p	u	w	x
ab		0.76	0.9	0.92	0.73	0.8	0.76	0.69
b	0.84		0.86	0.85	0.7	0.74	0.71	0.68
c	0.88	0.79		0.89	0.7	0.76	0.72	0.67
k	0.93	0.78	0.92		0.74	0.8	0.76	0.68
p	0.8	0.7	0.78	0.79		0.79	0.75	0.91
u	0.86	0.72	0.73	0.84	0.83		0.9	0.73
w	0.86	0.73	0.83	0.84	0.83	0.95		0.71
x	0.81	0.73	0.8	0.78	0.97	0.8	0.74	

Netsky

REFERENCES

[1] MySQL, http://www.mysql.com

[2] Esko Ukkonen, "Algorithms for approximate string matching", International Conference on Foundations of Computation Theory, Mar. 1986.

[3] Silvio Cesare, Yang Xiang, "Classification of Malware Using Structured Control Flow", in Proceedings of Australasian Symposium on parallel and Distributed Computing(AusPDC 2010), Brisbane, Australia, 2010.

[4] Reverse Engineering Compiler, http://www.backerstreet.com

[5] Silvio Cesare, Yang Xiang, Wanlei Zhou, ”Malwise–An Effective and Efficient Classification System for Packed and Polymorphic Malware”, IEEE Transactions On Computers, vol. 62, No. 6, Jun. 2013.

[6] AV-TEST, http://www.av-test.org

[7] Marius Gheorghescu, ”An automated virus classification system”, Virus Bulletin Conference, Oct. 2005.

[8] Weizhong Li, Adam Godzik, ”Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences”, Bioinformatics, vol. 22, pp.1658-1659, May. 2006.

[9] Vladimir Levenshtein, ”Binary Codes Capable of Correcting Deletions, Insertions and Reversals”, Soviet physics doklady, 1966.

[10] Offensive computing, http://www.offensivecomputing.net