International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 1 (February, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Pattern Matching Techniques for Metamorphic Virus Detection

Ankur Singh Bist[1], Dr. Anuj Sharma[2]
[1]Ph.D Scholar, SVU, Gajraula, [2]Visiting Faculty, SVU, Gajraula
[1]ankur1990bist@gmail.com, [2]draks1976@gmail.com

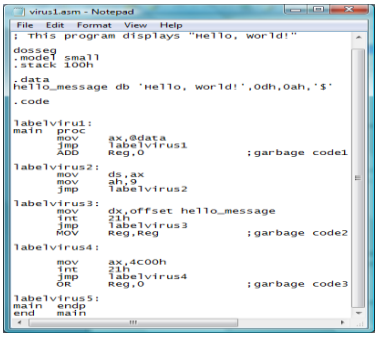
Abstract: Recent research work shows that pattern matching technique is widely used in computer virus detection. Viruses generated from kits like NGVCK are detected effectively using pattern matching approach. Our purpose is to examine various flavours of pattern matching approach in virus detection.

Keywords: Pattern Matching, Computer Virus, Obfuscation.

I. Introduction

Internet has become target of malicious codes due to its increasing use. Malicious codes are executable code and have the capability to replicate. It makes their survival strong. Viruses design and evolution attached with the area of programming. Similar to other computer programs viruses carry functions that are intelligent for providing protection in such a manner that detection remains not easy for virus scanner [1].

Viruses have to take various procedures of intellect for continued existence. That is why they may have complex encrypting and decrypting engines. These are the most frequent methods used by computer viruses in current scenario. They make use of these techniques to mask the antivirus and to adopt the certain environment for their expansion [2].


virus1asm - Notepad

```
File  Edit  Format  View  Help
; this program displays "Hello, world!"
dosseg
.model small
.stack 100h

.data
hello_message db 'Hello, world!',0dh,0ah,'$'
.code

labelvirus1:
main  proc
      mov     ax,@data
      jmp     labelvirus1
      ADD     Reg,0                 ;garbage code1
labelvirus2:
      mov     ds,ax
      mov     ah,9
      jmp     labelvirus2
labelvirus3:
      mov     dx,offset hello_message
      int     21h
      jmp     labelvirus3
      MOV     Reg,Reg               ;garbage code2
labelvirus4:
      mov     ax,4C00h
      int     21h
      jmp     labelvirus4
      OR      Reg,0                 ;garbage code3
labelvirus5:
main  endp
end   main
```

Figure 1: Assembly code of Virus File

Polymorphic viruses try to hide the decrypting module. More complex methods were developed enabling the virus designers to change the code of one virus file and make multiple morphed copies while maintaining its functionalities. These are the type of viruses which have the ability to mutate itself with the code changed but without changing its functionalities. Metamorphic virus can become a serious threat considering the fact that there can be thousands of variants of one virus file with their signature being totally different.

Metamorphic viruses transform its code in a specific manner very frequently and require to be prohibited. Their analysis will lead to evolve a framework where the overall process of detection will be bounded in specific outcomes of continuing evolving results. It is essential to make a distinction between replicating programs and its similar forms. Reproducing programs will not necessarily damage your system [3] [4] [5]. There is big fight between designers of virus and antivirus. The enhanced knowledge about the certain patterns, specifications can be designed. Various malicious codes can be evolved and incremented in well precise and efficient manner. For perfect identification of a metamorphic virus, identification routines must be written that can generate the essential instruction set of the virus code from the actual occurrence of the infection.

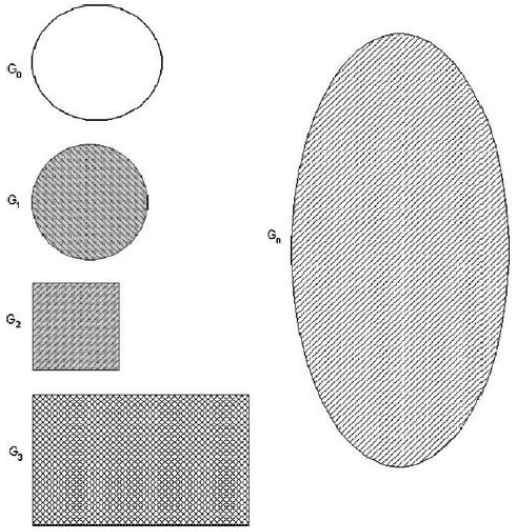


Figure 2: Analogy of Metamorphic Viruses

II. Malware Classification Using Pattern Matching

Code obfuscation is one of the important properties adopted by metamorphic viruses. The mutating behavior of metamorphic viruses is due to code obfuscation techniques. There are various code obfuscation techniques.

- Dead Code Insertion
- Variable Renaming
- Break And Join Transformation

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 1 (February, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

- Expressing Reshaping
- Statement Reordering

The analysis of opcodes has been used to study the malicious codes. The distribution of opcodes depicts the classifying feature that can be used for the purpose of detection. Pattern matching techniques have been used by authors to trace out classifying features between malicious codes and normal files. Some of frequently used techniques are listed as follows:-

- KMP Algorithm
- Aho Corasick
- Jaro Distance
- Cosine Similarity
- Pairwise Alignment
- Levenshtein Distance
- Damerau–Levenshtein Distance

Aho Corasick Algorithm

Let's look at the commented method which returns all the matches of the specified keywords:

```
// Searches passed text and returns all occurrences of any keyword
// Returns array containing positions of found keywords
public StringSearchResult[] FindAll(string text)
{
 ArrayList ret=new ArrayList(); // List containing results
 TreeNode ptr=_root;          // Current node (state)
 int index=0;                 // Index in text
// Loop through characters
 while (index<text.Length)
 {
  // Find next state (if no transition exists, fail function is used)
  // walks through tree until transition is found or root is reached
  TreeNode trans=null;
  while(trans==null)
  {
   trans=ptr.GetTransition(text[index]);
   if (ptr==_root) break;
   if (trans==null) ptr=ptr.Failure;
  }
  if (trans!=null) ptr=trans;
  // Add results from node to output array and move to next character
  for each(string found in ptr.Results)
    ret.Add(new StringSearchResult(index-found.Length+1,found));
   index++;
 }
 // Convert results to array
 return (StringSearchResult[])ret.ToArray(typeof(StringSearchResult));
}
```

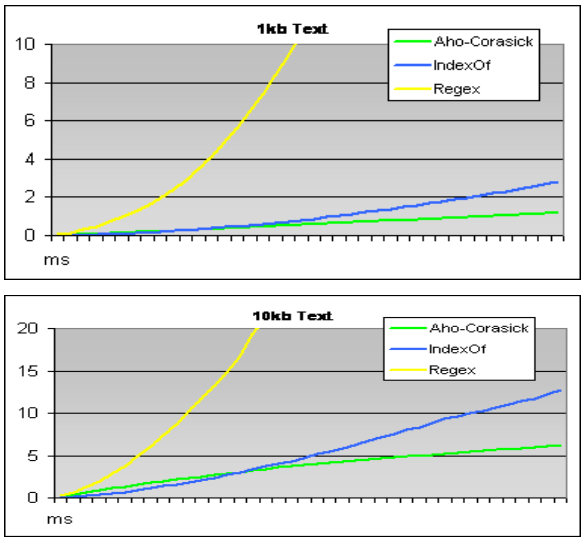


Figure 3: Two Charts Comparing The Speed Of The Three Described Algorithms - Aho-Corasick (Green), Indexof (Blue), And Regex (Yellow)

KMP Algorithm

KMP Search algorithm

```
int attempt(char *y, char *x, int m, int start, int wall) {
  int k;
     k = wall - start;
  while (k < m && x[k] == y[k + start])
    ++k;
  return(k);
}
     void KMPSKIP(char *x, int m, char *y, int n) {
  int i, j, k, kmpStart, per, start, wall;
  int kmpNext[XSIZE], list[XSIZE], mpNext[XSIZE], z[ASIZE];

  /* Preprocessing */
  preMp(x, m, mpNext);
  preKmp(x, m, kmpNext);
  memset(z, -1, ASIZE*sizeof(int));
```

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 1 (February, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

```
memset(list, -1, m*sizeof(int));
z[x[0]] = 0;
for (i = 1; i < m; ++i) {
  list[i] = z[x[i]];
  z[x[i]] = i;
}
  /* Searching */
wall = 0;
per = m - kmpNext[m];
i = j = -1;
do {
  j += m;
} while (j < n && z[y[j]] < 0);
if (j >= n)
 return;
i = z[y[j]];
start = j - i;
while (start <= n - m) {
  if (start > wall)
    wall = start;
  k = attempt(y, x, m, start, wall);
  wall = start + k;
  if (k == m) {
    OUTPUT(start);
    i -= per;
  }
  else
    i = list[i];
  if (i < 0) {
    do {
      j += m;
    } while (j < n && z[y[j]] < 0);
    if (j >= n)
      return;
    i = z[y[j]];
  }
  kmpStart = start + k - kmpNext[k];
  k = kmpNext[k];
  start = j - i;
  while (start < kmpStart ||
       (kmpStart < start && start < wall)) {
    if (start < kmpStart) {
      i = list[i];
      if (i < 0) {
        do {
          j += m;
        } while (j < n && z[y[j]] < 0);
        if (j >= n)
          return;
        i = z[y[j]];
      }
      start = j - i;
    }
    else {
      kmpStart += (k - mpNext[k]);
      k = mpNext[k];
    }
  }
 }
}
```

Jaro Distance

The Jaro distance d_j of two given strings S_1 and S_2 is:-

$$d_j = \begin{cases} 0 & if\ m = 0 \\ \frac{1}{3}\left\{\frac{m}{|s_1|} + \frac{m}{|S_2|} + \frac{m-t}{m}\right) & otherwise \end{cases}$$

Where:

- m is the number of *matching characters*.
- t is half the number of *transpositions*.

Two characters from S_1 and S_2 are *matched* only if they are the same and not farther than $\left\lfloor\frac{\max(|s_1|,|s_2|)}{2}\right\rfloor - 1$.

Jaro–Winkler distance uses a prefix scale p which provides more favorable scoring to strings that match from the start for a set prefix length l. Given two strings S_1 and S_2, their Jaro–Winkler distance d_w is:

$$d_w = d_j + (lp(1 - d_j))$$

where:

- d_j is the Jaro distance for strings S_1 and S_2
- l is the length of common prefix at the start of the string up to a maximum of 4 characters
- p is a constant scaling factor to trace out the adjusted score upwards for having common prefixes.

Cosine Similarity

Cosine similarity is used to determine the similarity between two vectors of an inner product space that

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 1 (February, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

calculates the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a conclusion of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is particularly used in positive space, where the outcome is clearly bounded in [0,1].

The cosine of two vectors can be derived by using the Euclidean dot product formula:

$$a.b = ||a||||b||cos\theta$$

Given two vectors of attributes, *A* and *B*, the cosine similarity, *cos(θ)*, is shown using a dot product and magnitude as:-

$$similrity = \cos(\theta) = \frac{A.B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i X B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2 X}\sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

The resulting similarity ranges from −1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating decorrelation, and in-between values indicating intermediate similarity or dissimilarity.

For matching the text, the attribute vectors *A* and *B* are usually the name frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison.

Pairwise Alignment Algorithm

Definitions

s1 = first sequence

s2= second sequence

|L|=length of sequence a

a_i=indicates the i^{th} symbol of sequence a

$a_{i...j}$= subsequence of a with indices i to j where a= $a_{i...|L|}$

scor(a, b) = score assigned to substituting symbols a with b

gap(n) = cost of adding one gap to sequence with n-1 gaps

F and G = matrix of size |s1|+1 * |s2|+1 (indices will be 0 based)

F(i, j) =optimal score for aligning $s1_{1...i}$ with $s2_{1...j}$

G (i, j) = number of subsequent gaps used to generate F(i, j).

Recursive definition of F and G for i, j≥0

G(i,0)=F(i,0)=0

G(0,j)=j

F(0, j)= $\sum_{n=1}^{j} gap(n)$ (the cost of aligning j gaps)

F(i, j)=max((F(i-1, j-1)+scor(s1i, s2i),F(i-1, j)-gap(G(i-1,j)),F(i, j-1)-gap(G(i,j-1))

if case1: G(i, j)=0

if case2: G(i, j)=G(i-1,j)+1

if case3: G(i, j)=G(i, j-1)+1

Pseudo Code

Initialize the first row in F and G: G (0, j)= j and F(0, j) = $\sum_{n=1}^{j} gap(n)$

For each row i , 1….|s1|

Initialize F(i, 0)=0 and G(i,0)=0

For each column j, 1….|s2|

(i-1, j-1), (i-1,j) and (i, j-1) for F and G are all known.

Calculate F(i, j) and G(i, j) using the recursive definition

Levenshtein Distance

Input – Two strings obtained from disassemble code.

```
int Levenshtein _Dis (char str [1…m], char str1 [1…n])
{
    Define int dis[0…m,0…n]
     for i from 0 to m
     dis[i,0] =i;
  for j from 0 to n
    dis[0,j] =j;
  for j from 1to n
 {
    For i from 1 to m
     {
  If ( str[i]  == str1[j]) then
   dis[i ,j] = dis[ i-1, j-1];
 else
   dis[i , j]=min
          {      dis [i-1, j]+1,
                 dis [i, j-1]+1,
                 dis [i-1, j-1]+1;
            }
```

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 1 (February, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

}

Damerau–Levenshtein Distance

The Damerau–Levenshtein distance between two strings a and b is given $d_{a,b}(|a|,|b|)$where:

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

. Cases in Damerau–Levenshtein distance:

- $d_{a,b}$(i-1, j)+1 corresponds to a deletion (from a to b).
- $d_{a,b}$(i, j-1) + 1 corresponds to an insertion (from a to b).
- $d_{a,b}$(i-1, j-1) + $1_{(ai \neq bj)}$ corresponds to a match or mismatch, depending on whether the respective symbols are the same.
- $d_{a,b}$(i-2, j-2) + 1 corresponds a transposition between two consecutive symbols.

III. Impact Of String Maching Techniques On Malware Classification

McGhee (2007) proposed a new approach for identifying metamorphic viruses which uses the fundamental concepts from bioinformatics. Pairwise alignment approach is used in bioinformatics to analyze the protein sequences. Same approach is used for analyzing the opcodes sequences of metamorphic viruses. This approach performed well for identifying metamorphic viruses. Scoring refinement and pre-processing are some aspects that are required to increase the effectiveness of this technique.

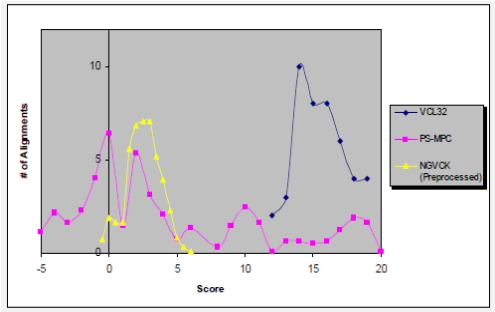


Figure 4: Score Distributions For VCL32, PS-MPC, NGVCK (Pre-Processed) Adjusted For A Sample Size Of 45 Alignments

Donabelle (2012) explained the structural entropy of metamorphic malware using edit distance. The experimental study provides the analysis that represents the capability of classifier to identify malware data generated by G2 and MWOR generator. The results that are obtained for various file sizes give detail structural distribution regarding identification of metamorphic viruses.

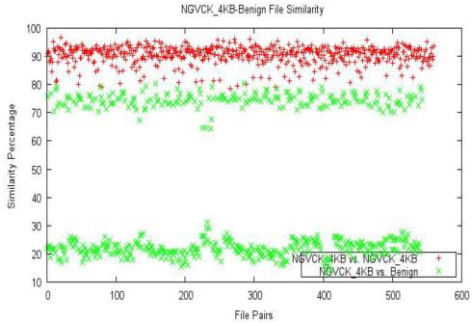


Figure 5: NGVCK (4 KB) Similarity

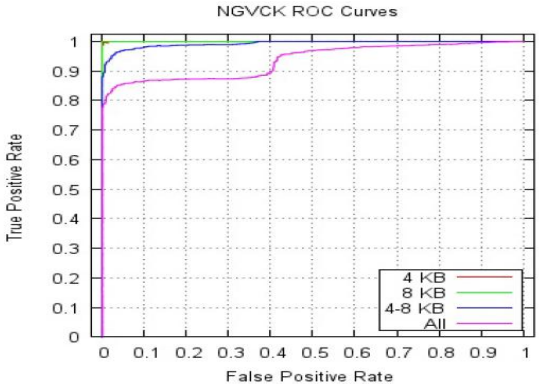


Figure 6: NGVCK ROC Curve

Cosine Similarity: Walenstein et al. explained about a method for searching database of programs for a match. The features compared are called n-perms obtained from disassembled code. Vilo approach is used for text retrieval matching using tf * idf term vector query matching algorithms. Meaningful features are traced using cosine similarity method.

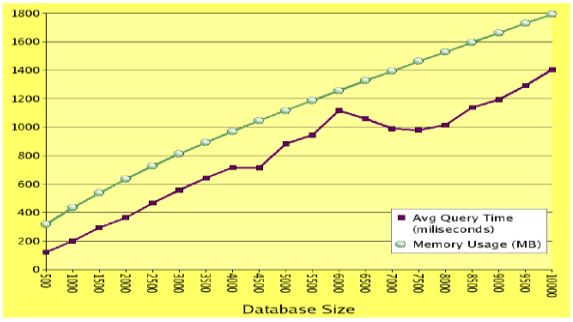


Figure 7: x axis denotes database size, red line with square represents average query time in milliseconds, and green line with circle represents memory usage in megabytes.

International Journal of Electrical Electronics & Computer Science Engineering
Volume 3, Issue 1 (February, 2016) | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Table 1

THRESHOLD	MEAN PRECISION	MEAN RECALL
.002	0.79	1.00
.100	1.00	1.00

Threshold, mean precision and mean recall is given above for the concerned database used.

Karim *et al.* explained about n perm technique and how this technique can be used to analyze typical body changing viruses like metamorphic viruses. Authors created two programs one for n gram and other for n perms and designed feature occurrence matrices after that cosine similarity method finally CLUTO is used to perform clustering.

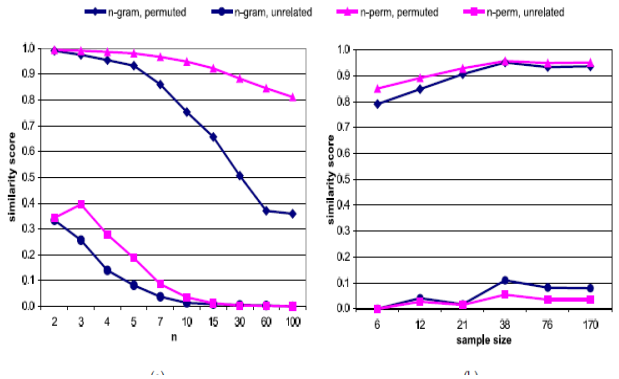


Figure 8: Similarity Scores Varying by N and Sample Size

Jared Lee presented a method based on structural entropy measurement to detect metamorphic malware. File segmentation and sequence comparison is used. Kolmogorov complexity is used to calculate compression ratio for malware.

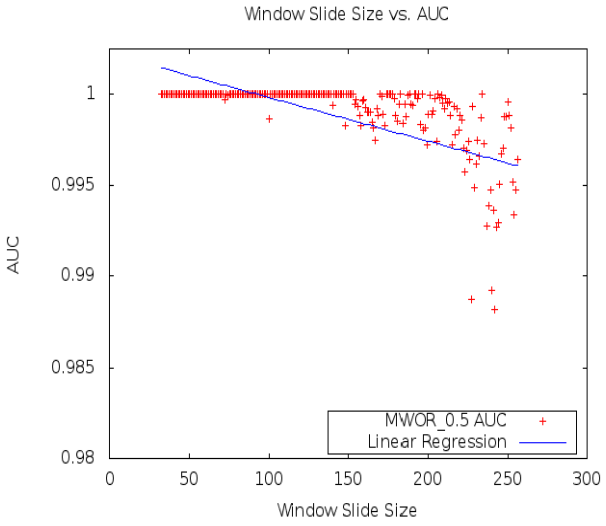


Figure 9: Window Slide Size vs AUC

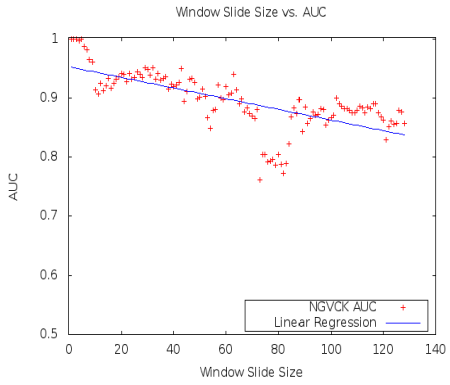


Figure 10: Window Slide Size Vs AUC

IV. CONCLUSION

Pattern matching tools are the powerful way to detect computer viruses. In this paper a detailed study is made to understand the impact of pattern matching techniques in malware detection especially in metamorphic virus detection. Literature study depicts the various dimensions of pattern matching techniques that are being explored by researchers in order to enhance its utility in malware detection.

V. REFERENCES

[1] J. Aycock, Computer Viruses and Malware, Vol 22, New York, NY, USA: Springer, pp. 5-32. 2006.

[2] H. Bidgoli, Handbook of information security, Wiley. 2006.

[3] F. Cohen, Computer Viruses. PhD thesis, University of Southern California. 1986.

[4] Bist, Ankur Singh. "Detection of metamorphic viruses: A survey." Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, 2014.

[5] Bist, Ankur Singh. "Classification and identification of Malicious codes." IJCSE. 2012.

[6] M. Mohammed and A. Lakhotia, 2003. A method to detect metamorphic computer viruses. The IEEE Computer Society's Student Magazine, Vol. 10(1).

[7] http://www.codeproject.com/Articles/12383/Aho-Corasick-string-matching-in-C, Last visited 23 March 2015.

[8] C. Charras, T. Lecroq, J. D. Pehoushek , 1998, A very fast string matching algorithm for small alphabets and long patterns, in Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching , M. Farach-Colton ed., Piscataway, New

Jersey, Lecture Notes in Computer Science 1448, pp 55-64, Springer-Verlag, Berlin.

[9] L. Graham Giller, 2012. "The Statistical Properties of Random Bitstreams and the Sampling Distribution of Cosine Similarity".

[10] S. McGhee, 2007. Pairwise alignment of metamorphic viruses, Department of Computer Science, San Jose State University.

[11] Donabelle, B.2011. Structural Enotropy of Metamorphic Malware. Department of Computer Science, San Jose State University.

[12] Md. E. Karim, A. Walenstein and A. Lakhotia, Malware phylogeny generation using permutation of code, Software research laboratory, University of Louisiana Lafayette..

[13] A. Walenstein, M. Venable, M. Hayes, C. Thompson and A. Lakhotia, Exploiting similarity between variants to defeat malware, Software research laboratory, University of Louisiana Lafayette.

[14] L. Jared, 2013, Compression based analysis of metamorphic malware, Master's Projects. pp. 329.