A Hybrid Technique for Complete Viral Infected Recovery

Pawut Satitsuksanoh, Peraphon Sophatsathit, and Chidchanok Lursinsap
Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics, Faculty of Science
Chulalongkorn University
Bangkok, Thailand
nnumbkrub@yahoo.com, peraphon.s@pioneer.netserv.chula.ac.th,
lchidcha@pioneer.netserv.chula.ac.th

Abstract

This research proposes a hybrid technique for computer virus detection and recovery. We made use of the well-established BWT to pinpoint where the infection was located. To insure perfect detection, the CRC technique was supplemented. In the mean time, the original uninfected code was analyzed to obtain necessary unique identifications, whereby recovery process can be carried out directly with reference to these unique identifications. The proposed technique was gauged against a couple of commercial virus software and found to perform its task to perfection.

Keyword computer viruses, virus detection and disinfection, BWT compression, data integrity check, information security.

1. Introduction

Anti-virus software today is fairly sophisticated, but virus writers are often a step ahead of the software. New computer viruses are constantly being released which the current anti-virus software cannot recognize. Most anti-virus systems are still based on scanning detection using virus signature because of their very low false alarm [1,2]. To get a new virus signature, the anti-virus researcher has to analyze the infected code of a host file in order to extract the specific pattern of a particular virus before releasing a new updated signature file. This process may take quite a long time for complicated coding viruses for instance, armored virus, polymorphic or metamorphic virus. This is a main drawback of using signature based virus detector. We are interested in not only the problem of detecting virus but also the problems of disinfecting and cleaning virus from the target program. There are many kinds of virus which destroy or replace target files. Existing commercial anti-virus systems cannot recover back the healthy program from these kinds of infection. The only possible solution is to delete the infected file and reinstall from the previously back up file. From the previously stated drawbacks, we have proposed a framework to create a file archive together with message digest for virus, change detection, file recovery, and virus cleaning. Details of the proposed technique will be described in subsequent sections. This paper is organized as follows. Section 2 describes background in computer virus. Some fundamental techniques are described in related work of Section 3. The proposed technique is elucidated in Section 4, along with experimental results in Section 5. Some final thoughts are given in Section 6.

2. Background

In this research, we focus on real computer viruses which infect or change the contents of files. These viruses can be classified by the way they operate on the host file.

2.1 Classification of virus infection techniques

Computer viruses can be classified according to different aspects such as target format, behavior of each virus, payload type, etc. A popular technique is based on infection techniques [1,2,3] as follows:

2.1.1 Overwriting viruses. This infection technique simply overlays part of the existing target code with the virus own copy. The size of the infected files may increase or decrease if it is completely replaced by the virus code. The infected file may have the same size as the original one if it is partly replaced with viral code. Overwriting viruses cannot be disinfected from a system by the

existing anti-virus program. Infected files must be deleted from the disk and restored from backups.

2.1.2 Adding viral code: appenders and prependers. The technique gets its name from the location of the virus body, which is added at the beginning or the end of the target program. This method will inevitably increase the size of the infected file unless a stealth technique is applied.

2.1.3 Code interlacing infection or hole cavity infection. This infection technique typically does not increase the size of the infected target. The cavity virus overwrites a portion of the file to safely store the virus code. It typically overwrites areas of files that contain zeros in binary files or code areas that have been allocated by the compiler but only very partially used by the code itself**.**

2.1.4 Companion viruses. This infection technique is quite different from all previously mentioned techniques. The target code is not modified, thus preserving the code integrity. The companion virus operate as follows. The viral code identifies a target program to attack and create an additional file, which is somehow linked to the target code to be executed in place of the target file.

2.2 Anti-virus techniques

The most efficient modern anti-virus applications have combined several different techniques [1,2,3] which are briefly described below.

2.2.1 Searching for virus signature. This technique searches for any known sequence of bits which distinguishes a particular infected program from other programs. This technique is still used by most commercial anti-virus programs because it can detect known viruses efficiently. However, this technique fails to handle unknown or armored viruses such as polymorphic viruses or metamorphic virus. A major drawback of this technique is that it must keep the virus signature database up-to-date and secured during distribution and use.

2.2.2 Spectral analysis. This technique statistically analyzes instructions of a given program to find subsets of unusual instructions or contain feature specific to viruses. Thus, this technique may cause many false alerts. Fortunately, the advantage of this technique is that some unknown viruses may be detected by incorporating into other known techniques.

2.2.3 Heuristic analysis. This technique uses rules and strategies to study how a program behaves. The purpose is to detect potential virus activities or behavior. The advantage and drawback of this technique are similar to spectral analysis which can detect unknown viruses but produce more false alerts.

2.2.4 Activity monitoring. This technique monitors various activities of viral programs by being memory-resident to detect and stop any potential suspicious activities. This technique may sometimes succeed in both detecting unknown viruses and avoiding infections. The drawbacks are producing more false alert, requiring frequent update of virus behavior database, and degrading system performance as it operates in real-time mode.

2.2.5 Code emulation. This technique utilizes a virtual machine to mimic code execution under CPU and memory management systems. Thus, infected code is simulated in the virtual machine of the scanner having no actual virus code executed by the real processor. This technique can detect encrypted, polymorphic, and metamorphic viruses at the expense of computer resources and time.

2.2.6 File integrity check or change detection. This technique aims at monitoring and detecting any modification of sensitive files such as executables, documents, etc. Traditionally for each file, the file digest is computed with the help of either hash function such as MD5 or SHA-1, or cyclic redundancy codes (CRC) [4]. Our proposed technique is in this category. There is a known issue of using CRC for the purpose of virus detection or file integrity check is vulnerability to be exploited by the virus writer [4,5]. This is not the case for our proposed technique because CRC is used as the supplementary check in the message digests.

2.3 BWT compression

Our proposed technique is primarily based on Burrows-Wheeler Transformation (BWT) [6]. BWT is the heart of a compression algorithm. The BWT itself is not a compression technique but permutates the original data to be more compressible for further processing.

The first step of BWT compression is to take a string S of N symbols S[0], S[1], …, S[N-1] and construct the N rotation strings such that:
S[0], S[1], .., S[N-2], S[N-1]
S[1], S[2], .., S[N-1], S[0]
…

S[N-1], S[N-2], …,S[1], S[0]
A table of N rows is formed and sorted lexicographically. The output of transformation is the last column and the index which is called r_index in this paper. The attribute of r_index is the reverse BWT. An example of the transformation over the string, 'ubuntu' is shown in Figure 1.

S = 'ubuntu' N= 6	u b u n t u b u n t u u u n t u u b n t u u b u t u u b u n u u b u n t
Row 0 b u n t u u 1 n t u u b u 2 t u u b u n * 3 u b u n t u 4 u n t u u b 5 u u b u n t	TM = 'uunubt' r_index = 3

Figure 1. Example of performing BWT over string S = 'ubuntu'.

The transformed block is further processed by Move-to-Front (MTF) and Run Length Encoding (RLE) function before it is compressed by the Compression Module using entropy encoding techniques such as Huffman encoding or Arithmetic encoding. Details on how it works can be found in [6,7,8].

3. Related work

Because of the limitation in detecting unknown computer viruses, many researchers have proposed virus detection techniques based on biologically inspired techniques [9,10,11,12,13]. Most of them refer to the great ability of human immune system in protecting human body from unknown pathogen like biological viruses and propose an artificial immune system to protect the computer from computer viruses. For example, Lee, et al. [9] work on artificial immune based virus detection system that can detect unknown viruses. Their work is based on self and nonself strings defined previously in Forrest's research [13]. Other researchers [14,15] proposed computer viral detection techniques based on artificial neural networks. Their techniques do not required signature for detecting unknown viruses. Some recently researches emphasize on detecting hard to detect metamorphic computer viruses [16,17], introducing the term "virus localization" [18]. The underlying principle of this research is a multiple cryptography hashing technique to locate areas within the infected file.

None of previously stated researches have suggested any approach to heal the infected code, which differ from our proposed approach.

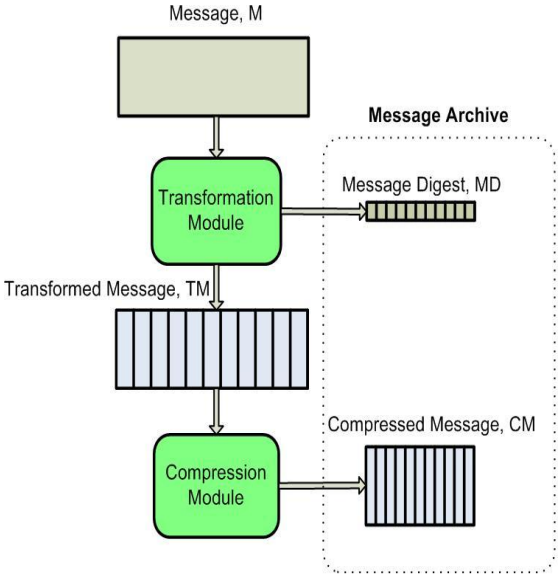

Figure 2. Archival construction diagram.

4. Proposed technique

From the preliminary experiment, we found that whenever the content of the message changed, the r_index would change as well. Even though the result of using indices from BWT process was quite good, these indices alone could not be used as a hash function for the integrity checking. Therefore, CRC-32 [19] are applied to supplement these indices, serving as the basis for our proposed technique. Figure 3 shows the overview of error detection and recovery process.

The proposed technique consists of three processes namely, archival construction process, error detection process, and recovery process, which are described below.

```
Notation
OF              Original file
MF              Message digest file
CF              Compressed file
t()             Transformation function
(TMi, r_indexi) Output of transformation
                function, the first tuple is
                a transformed message block
                 the second tuple is the
                 association index at index i
crc()           CRC checksum computation
                function
CRCi            CRC checksum at index i
compress()      Compression function
Mi              Message block at index i
CMi             Compressed Message block at
                index i
block_size      block size
N               number of blocks
Di              Digest block at index i
A | B           | is defined as a
                concatenation operator

Archival Construction Process

Input : OF, block_size

1   N ß sizeof(OF)/ block_size
2   OF = M0M1...MN-1
3   open(MF) for write
4   open(CF) for write
5   For i ß 0 to N -1 do
6       (TMi, r_indexi)  ß  t(Mi)
7        CRCiß  crc(TMi)
8        Di ß r_indexi | CRCi
9        write(MF,Di)
10       CMi ß compress(TMi)
11       write(CF,CMi)
12  end_do

Output : CF, MF
```

Figure 3. Pseudocode for archival construction precess.

4.1 Archival Construction Process

This is the first process that is responsible for rearranging, compressing, and computing necessary information for message archival purpose. As shown in Figure 2 , a message (or a file) is passed to this process where a compressed message along with the message encoder of the original message is returned. The process can be described in pseudo code as shown in Figure 3. A file and specified block size (block_size) are passed to the Transformation Module. The entire message is implicitly chopped down to N blocks. Each block is transformed by the BWT algorithm and the corresponding CRC checksum is computed. The output of this stage is a blockwise message digest(D, digest block), which is the combination of r_index and CRC checksum. The transformed message block is further processed by the Compression Module in Figure 2 which associates to compression function in the pseudocode. The compression function can be implemented using MTF and RLE function before

```
Notation
MF'     Message digest file of the
        suspected file
D'i     Digest block of suspected file at
        index i
Ns      Number of blocks of suspected file
ErrB    Used for keep error block number
        and associated digest block
ErrLoc  Error locating file

Error Detection Process

Input : MF, MF'

1  MF = D0D1…DN-1
2  MF' = D'0D'1…D'Ns-1
   open(ErrLoc)for write
3  if (N > Ns)
4    for i ß 0 to Ns -1 do
5         if (Di != D'i )
6             ErrB ß i | Di
7             write(ErrLoc,ErrB)
8    end_do
9    for j ß i to N -1 do
10         ErrB ß j | Dj
11         write(ErrLoc,ErrB)
12   end_do
13 else
14    for i ß 0 to N -1 do
15        if (Di != D'i )
16           ErrB ß i | Di
17           write(ErrLoc,ErrB)
18    end_do

Output : ErrLoc
```

Figure 4. Pseudocode for error detection precess.

it is encoded in the final state by entropy encoding techniques such as Huffman encoding or Arithmetic encoding. In each iteration, the blockwise message encoder and compressed block separately form the message digest file and compressed file, respectively.

The message block size can be arbitrary selected to discourage any guess work of the virus writers in decoding attempts. In addition, both message digest file and compressed file can be physically separated from the working file for subsequent error detection and recovery processes, making malicious decoding virtually impossible.

4.2 Error detection process

The main purpose of this process is to detect and locate error blocks in the message (or the file). The outputs of this process are the number of error blocks, the information to be used in decompression, and reverse transformation of the specified message block, all of which will be used in subsequent processing.

The procedures of this process are described in Figure 4. The input of this process are the message digest file of the original file and the message digest file of the suspected file. The message digest

```
Notation
IF              Infected file
Nerr            Number of error blocks
Pos             Used for keep error block
                number
uncompress()    Uncompressing function
reverse_t()     Reverse transformation function
replace()       Replacement function
spilt()         Spilt function which return a
                pair of variable (a,b)

Recovery Process

Input : IF, CF, ErrLoc

   open(ErrLoc) for read
1  For i ß 1 to Nerr do
2       ErrB ß read(ErrLoc)
3       (Pos,D) ß spilt(ErrB)
4       (r_index,CRC) ß spilt(D)
5       TM ß uncompress(CMPos)
6       M ß reverse_t(TM, r_index)
7       replace(IF, M, Pos)

Output : Disinfected file
```

Figure 5. Pseudocode for recovery process.

of the suspected file can be computed by using the same procedure as in archival construction process excepts that the compressed form of the suspected message is not required. The digest block of the original and suspected files will be compared one by one. If they are not equal, the block number and digest block, which is a pair of r_index and CRC checksum, will be recorded into ErrLoc file. Three possibilities to be considered of the number of digested blocks of the original file and the suspected file are greater, less, or equal.

4.3 Recovery process

This process will recover the original message from file archive using information from the previous process. The procedure of this process is given in Figure 5. The inputs for this process are infected file, CF, and ErrLoc. The number of error blocks is retrieved from the ErrLoc file. For each iteration, reverse transformation of the uncompressed block will replace the specified error block without having to go though the entire file. Finally, the original file or message is restored . Figure 6 shows an simple example of the process.

5. Experimental results

The experiments were conducted in two phases, namely, the preliminary experiment and the proposed method experiment.

5.1 Preliminary experiment

In preliminary experiment phase, the indices which derived from forward BWT were investigated to locate any discrepancies caused by content modification. We wanted to explore the pattern of change indicated by these indices as the contents were altered.

Table 1. Show the result of using r_index as a change detector.

File name	Change Detection Rate (%)				
	1 Byte	2 Bytes	4 Bytes	8 Bytes	16 Bytes
bib	45.67	49.83	71.33	89.83	97.83
bmp1.bmp	35.75	41.50	36.00	48.50	52.50
bmp2.bmp	58.83	59.17	62.67	66.33	67.67
book1	46.17	66.67	72.50	89.83	93.83
book2	42.17	59.00	73.67	89.00	95.83
geo	26.00	41.67	51.67	55.83	65.17
news	46.00	59.00	76.67	84.17	89.17
obj1	26.00	35.00	39.67	65.33	80.33
obj2	26.33	30.17	52.33	69.67	78.67
paper1	68.80	86.80	93.00	96.60	98.60
paper2	68.67	84.67	94.33	96.67	98.17
paper3	76.75	85.00	95.50	99.50	99.75
paper4	29.00	45.67	66.33	93.67	100.00
paper5	69.00	89.33	85.33	91.33	92.67
paper6	30.25	53.75	79.00	93.00	99.25
pic	57.67	58.33	54.17	60.17	61.17
progc	40.00	43.50	73.00	88.00	98.75
progl	33.00	58.20	73.60	86.00	92.40
progp	64.50	76.75	85.75	83.50	91.25
sendmail	41.83	64.50	74.00	79.50	89.67
tcpdump	30.17	48.83	63.67	78.33	85.50
trans	74.50	84.00	84.67	86.83	90.33

The Calgary corpus [20] and four other files were selected to furnish an extensive file type coverage in the test set. Four additional files are added, consisting two Microsoft bitmap images and two unix program files, namely, bmp1.bmp, bmp2.bmp, sendmail.sendmail, and tcpdump, respectively. Numerous test sets were generated by arbitrarily selecting a pseudo random location to seed contiguous change of various sizes from 1 bit to 16 bytes in different blocking volumes. The results of the experiments are shown in Table 1. It was observed that, in most cases, as the size of seeded contiguous change increased, the indices that indicated content change also increased. Nevertheless, certain singularities remained undetected, such as similar bit patterns or coincidental computed values, etc. Such caveats were compensated by additional CRC supplement that yielded 100% correct detection.

S_1 = 'ubuntu', block size = 3 M_0 = 'ubu', M_1 = 'ntu' TM_0 = 'uub', r_index_0 = 1 TM_1 = 'unt', r_index_1 = 0	S_2 = 'ubantu' M_0 = 'uba', M_1 = 'ntu' TM_0 = 'bua', r_index_0 = 2 TM_1 = 'unt', r_index_1 = 0
Error block is located and it can be decompressed located block, reversed transform and derived 'ubu'. S_2 = 'ubantu' replaced with 'ubu' at first block (0) S_2 = S_1= 'ubuntu'	

Figure 6. Example of error detecting and recovering

5.2 Proposed method experiment

The same testing sets were tested in the error detection process that every error block can be detected. The results are shown in Figure 7. For a set of selected block sizes, the size of compressed file of file archive is shown in Table 2.

Table 2. Show size of considered files after apply with BWT compression.

File name	File size (Bytes)		Compression Ratio
	Original File	Compressed File	
Bib	111,261	29,567	3.76
bmp1.bmp	67,854	17,431	3.89
bmp2.bmp	1,497,206	18,944	79.03
book1	768,771	275,831	2.79
book2	610,856	186,592	3.27
Geo	102,400	62,120	1.65
News	377,109	134,174	2.81
obj1	21,504	10,857	1.98
obj2	246,814	81,948	3.01
Paper1	53,161	17,724	3.00
Paper2	82,199	26,956	3.05
Paper3	46,526	16,995	2.74
Paper4	13,286	5,529	2.40
Paper5	11,954	5,136	2.33
Paper6	38,105	13,159	2.90
Pic	513,216	50,829	10.10
Progc	39,611	13,312	2.98
Progl	71,646	16,688	4.29
Progp	49,379	11,404	4.33
sendmail	3,859,419	1,375,653	2.81
tcpdump	448,056	207,949	2.15
Trans	93,695	19,301	4.85

Note from Table 2 that compressibility of the original file (or message) depends primarily on file type as observed from the resulting compression ratio. Additional major benefits from the proposed approach are (1) content verification of suspicious files (or messages) can be carried out in compressed from without any decompression overhead; (2) off-line vital archives preserve the integrity of the original information, thereby easing the recovery process considerably

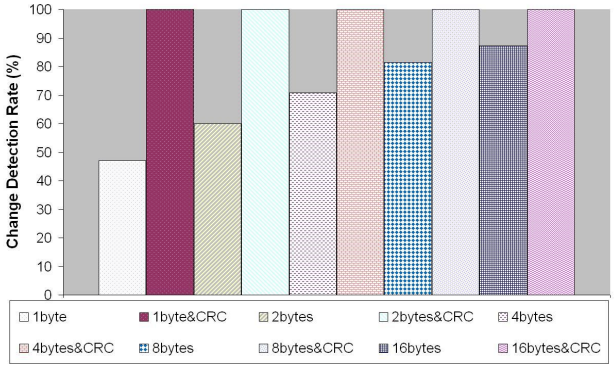

Figure 7. The graph shows using CRC as the supplementary.

5.3 An experiment over real computer viruses

A collection of computer viruses were cultured in a controlled environment. Various virus types infected on target files were analyzed, namely, overwriting virus, appending virus, prepending virus, and companion virus. Two well-known commercial anti-virus software were deployed along with the proposed technique. They are Avira Antivir Personal and ESET NOD32 Antivirus. Table 3 summarizes the results from real computer virus infection. Six major types of viruses were deployed, namely, appending, prepending, adding, hole cavity, overwriting, and companion viruses. Four categories of viruses that were shown to be detrimental are hole cavity, adding, prepending, and overwriting viruses. All of which required 100% replacement owing to total infection. The rest were relatively typical of virus infection with one exception, i.e., MRT.exe having 0.06% replacement. This was resulted from infection only in small number of blocks in a large file. Note that only hole cavity virus (PING.EXE) that yielded the same file size after infection. At any rate, the proposed technique successfully recovered the infected files to their original status. No commercial software could match the performance by any measures.

6. Conclusion

This research proposes a practical, yet efficient method for virus detection and virus disinfection in a message or a file. The proposed method not only is able to pinpoint the location of error, but also perform a perfect error recovery. The approach utilizes the fast BWT algorithm complemented by the CRC technique to arrive at a 100% damage repair. Moreover, the proposed method offers a number of security-tight features such as 1) off-line compressed archives of vital information 2) parameterized block size and index to preclude any illegal modifications, despite known algorithms, and 3) low computation overheads as related parameters can be made available during verification and required to be updated occasionally. We shall extend the proposed method to cover randomized error seeding and gauge the performance of our proposed method. We envision that the proposed method can be incorporated in other research and development areas, in particular, commercialization as the method is straightforward to implement on available technology.

Table 3. The summarization of virus detection and disinfection by using the proposed technique.

Infected File Name	original file size (bytes)	file size after infected (bytes)	Virus Name	Virus type	Proposed Disinfection (% of recovery)	% of file replacement	Commercial Anti-virus Software suggestion
setup.exe	116880	120,464	Win32/Basket.A	Appending virus	100%	16%	delete or quarantine
WAVTOASF.EXE	111632	115216	Win32/Basket.A	Appending virus	100%	17%	delete or quarantine
DotNetInstaller.exe	5632	125440	Win32/BCB.A	companion virus (the original file become DotNetInstaller.exe.exe)	100%	100%	delete or quarantine
notepad.exe	69120	8192	Win32/Belod.A	companion virus (the original file become notepad.dat)	100%	100%	delete or quarantine
DTAC_Edge.doc	24064	28672	W97M/Deij.A	Macro Virus (Overwriting Virus)	100%	100%	delete or quarantine
smiley.doc	41472	11264	Wm/Over.A	Macro Virus (Overwriting Virus)	100%	100%	delete or quarantine
ChCfg.exe	49152	53328	Win32/Cabanas.3014.A	Appending virus	100%	30%	delete or quarantine
MRT.exe	23635392	23638545	Win32/Cabanas.3014.A	Appending virus	100%	0.06%	delete or quarantine
Foxit Reader.exe	5713920	6053916	Win32/HLLP.Shodi.I	Prepending virus	100%	100%	delete or quarantine
UpdatPnP.exe	128512	169984	Win32/Neshta.A	Appending virus	100%	38%	delete or quarantine
MOM.exe	49152	666,624	Win32/Muce.A	Adding Viral Code	100%	100%	delete or quarantine
PING.EXE	24576	24576	Win95/CIH-2563.B	Hole Cavity virus	100%	60%	delete or quarantine

7. References

[1] P. Szor, "*The Art of Computer Virus Research and Defense"*, Addison-Wesley Professional, Boston, MA (2005).

[2] E. Filiol, "*Computer viruses: from theory to applications*", Springer-Velag France 2005.

[3] J. Aycock, "*Computer Viruses and Malware*", Springer 2006.

[4] D. Varney, "Adequacy of Checksum Algorithms for Computer Virus Detection", *Proceedings of the 1990 ACM SIGSMALL/PC Symposium on Small Systems*, pp. 280-282, March 28-30, 1990.

[5] B. Maxwell, D. R. Thompson, G. Amerson, and L. Johnson, "Analysis of CRC methods and potential data integrity exploits," *Proc. Int'l Conf. Emerging Technologies*, Minneapolis, MN, Aug. 25-26, 2003.

[6] M. Burrows, D.J. Wheeler, "A block sorting data compression algorithm", Tech. Report, Digital System Research Center, 1994.

[7] M. Nelson, "Data compression with the Burrows-Wheeler transform", *Dr. Dobb's J. Softw. Tools*, Volume 21, issue 9, pp. 46-50, 1996.

[8] P. Ferragina, R Giancarlo, and G. Manzini, "The engineering of a compression boosting library: theory vs practice in BWT compression", *LNCS*, vol. 4168, pp. 756-767, 2006.

[9] H. Lee, W. Kim,and M. Hong, "Artificial Immune System against Viral Attack", *ICCS2004*, *LNCS* 3037, pp. 499-506, Springer-Verlag Berlin Heidelberg 2004.

[10] K. S. Edge, G. B. Lamont, and R. A. Raines, "A Retrovirus Inspired Algorithm for Virus Detection & Optimization", *GECCO'06*, pp. 103-110, July 8-12, 2006.

[11] J. O. Kephart, "A biologically inspired immune system for computers", *Proceedings of the Fourth International Workshop on Synthesis and Simulation of Living Systems* (R. A. Brooks and P. Maes, eds.), pp. 130-139, Cambridge, MA: MIT Press, 1994.

[12] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, and S. R. White, "Biologically inspired defenses against computer viruses", *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (IJCAI'95), (Montreal, PQ), pp. 985-996, Morgan Kaufman, 1995.

[13] S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer", *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA: IEEE Computer Society Press, pp. 202-212 (1994).

[14] B. Zhang, J Yin, W. Tang, J. Hao, and D. Zhang "Unknown Malicious Codes Detection Based on Rough Set Theory and Support Vector Machine", *International Joint Conference on Neural Networks*, July 2006.

[15] I. S. Yoo and U. Ultes-Nitsche ,"Non-signature based virus detection" *Journal in Computer Virology*, Volume 2, Number 3, pp. 163-186 (2006).

[16] M. Webster and G. Malcolm, "Detection of metamorphic computer viruses using algebraic specification", *Journal in Computer Virology*, Volume 2, Number 3, pp. 149-161 (2006).

[17] W. Wong and M. Stamp, “Hunting for metamorphic engines”, *Journal in Computer Virology*, Volume 2, Number 3, pp. 211-229 (2006).
[18] G. D. Crescenzo and F. Vakil, “Cryptographic hashing for virus localization”, *Proceedings of the 4th ACM workshop on Recurring malcode*, November 2006.
[19] P. Koopman, “32-bit cyclic redundancy codes for Internet applications”, *Intl. Conf. Dependable Systems and Networks* (DSN), Washington DC, pp.459-468, 2002.
[20] The Calgary corpus may be downloaded from ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus