A Memory Symptom-based Virus Detection Approach

Hsien-Chou Liao and Yi-Hsiang Wang

(Corresponding author: Hsien-Chou Liao)

Department of Computer Science and Information Engineering, Chaoyang University of Technology
168 Gifeng E. Rd., Wufeng Taichung County, Taiwan 413, R.O.C. (Email: hcliao@mail.cyut.edu.tw)

(Received Aug. 5, 2005; revised and accepted Sept. 12, 2005)

Abstract

The widespread use of the Internet has caused computer security to become an important issue. Currently, anti-virus software is the primary mechanism that prevents computers from the damage of viruses. Such a mechanism relies on the update of virus patterns (or signatures) to detect new viruses. However, serious damage is usually caused before the update occurs. In addition, a few modification of the same virus can pass the pattern matching. This is one reason that the quantity of new viruses has exceeded 600 per month. This situation has also caused inefficiency in virus scans. To overcome the above problems, a new memory symptom-based approach is proposed in this paper. This idea comes from how diseases are diagnosed in real life. Doctors diagnose diseases based on the symptoms of a patient, such as a fever, a cough, etc., rather than based on the type of virus. Similarly, the program execution requires the usage of computer resources, such as CPU, memory, network, etc. We define the usage of a resource as a "symptom" of the program. Viruses can be detected according to their symptoms. In this paper, we focus on the memory symptom. The memory symptom of an unknown program is sampled, encoded, and matched with those of sample programs. Then a certainty factor (CF) value is computed to represent the possibility that the unknown program is a virus. In the experimental study, 109 test programs were detected. According to the analysis of the confusion matrix, a true positive rate can be as high as 97 percent, and a false positive rate can be 13 percent while the unknown rate is only 18 percent. This shows that the memory symptom-based approach is effective for virus detection.

Keywords: Computer security, memory symptom, virus detection

1 Introduction

The widespread use of the Internet and the popularity of computers have caused computer security to become an important issue. For many reasons, it is important to prevent the computers from the damage of viruses. Currently, anti-virus software is the most frequently used mechanism against viruses. Anti-virus software relies on a set of feature instructions, called patterns or signatures. Patterns are made from known viruses. A new virus can be detected only when its pattern is extracted and included in the pattern database of an anti-virus software. The period from the breakout of a virus to the extraction of its pattern is called zero-day. Serious damage is usually done on zero-day. For example, the famous Code-Red worm infected 359,000 computers within 14 hours [1]. The effect of the Slammer worm was even more damaging, infecting 75,000 computers within only ten minutes [4]. This is one problem of pattern-based anti-virus software.

In addition, a few modifications of the same virus can pass the pattern matching of anti-virus software. This is one reason that the quantity of new viruses has exceeded 600 per month. The increase of viruses also increases the number of patterns, causing the efficiency of virus detection to become worse and worse.

Although pattern-based anti-virus software is commonly used, three problems still exist:

1) The damage during zero-day is inevitable. Virus detection relies on the update of patterns. Serious damage may be caused during zero-day [7]. Some viruses, for example, the Netsky virus, can even cause network bandwidths to be occupied in order to prevent the update of patterns. The longer zero-day is, the more serious the damage can be.

2) Virus scanning is inefficient. The number of patterns increases very fast. This causes the time required for scanning viruses to be longer than ever before.

3) Viruses easily mutate. For example, there were more than 26 mutations of the Netsky virus within two months. This problem causes an increase in the number of patterns. This increase in the number of patterns means that an anti-virus software must update

patterns frequently. Too many patterns cause inefficiency in the scanning of viruses. This is a vicious circle.

In order to overcome the above problems, a memory symptom-based virus detection approach is proposed in this paper. It differs from the pattern-based approach. The idea comes from understanding how diseases are diagnosed in real life. Doctors diagnose diseases based on the symptoms that a patient has, such as a fever, a cough, etc., rather than the type of virus. Although a disease may be caused by many kinds of viruses, its symptoms are almost the same. Similarly, if a virus can be detected based on the symptoms it causes, regardless of how the virus mutates, it can still be detected.

In this paper, we mainly focus on the memory symptom of a program. The memory is consumed after the execution of a program. The memory symptom is the timely memory usages of program execution. Memory usage is sampled in fixed intervals and analyzed to detect whether a program is a virus or not. Basically, it is easy to generate a new viruses but difficult for them to cause new symptoms. The symptom-based approach is a new approach for detecting viruses.

The rest of the paper is organized as follows. Section 2 presents some related works. Section 3 explains further what the memory symptom is. Section 4 presents the proposed symptom-based approach. Section 5 presents the experimental study. And, finally, section 6 gives the conclusion of this paper.

2 Related Works

Many studies were conducted on how to detect computer viruses. For example, Lee et al. proposed a Virus Instruction Code Emulation (VICE) system [3]. The system provides a virtual environment for the execution of suspicious programs. A series of executed instructions are matched with those in a well-defined knowledge base in order to judge whether the program is a virus or not. 3,562 viruses were used in the experiment. 74.5 percent of the viruses were detected successfully.

In addition, Okamoto et al. proposed a novel approach for recovering infected files [6]. A set of heterogeneous agents is installed in different computers connected via network. Files are stored in more than one computer. An agent compares files with the same files in different computers. A file is judged to be infected when the same file in two computers is different. The infected file is recovered from the uninfected one. In other words, virus detection is the backing up and restoring of files. However, the approach cannot detect infected files without any modifications on themselves but their execution may cause malicious actions. For example, the execution of a file sends a large amount of e-mails.

Wang et al. used data mining technology incorporating the decision tree and Bayesian network to detect viruses [9]. A set of feature instructions is mined from a set

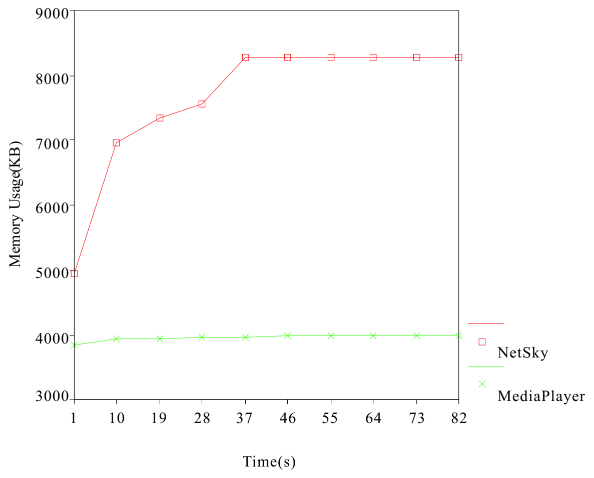


Figure 1: Memory usages of two programs

of viruses. The feature instructions are learned by the decision tree and Bayesian network. Then the decision tree and Bayesian network are used for detecting viruses. The accuracy rate was 79.4 percent in their experimental study.

Schultz et al. also utilized data mining technology for virus detection [8]. A tool, Hexdump, is used to convert a binary file into a series of hex codes. A sequence of bytes is extracted from the hex codes. Then the byte sequence is learned using the multi-naive Bayes algorithm. The learning results are used for virus detection. The experimental results showed that the accuracy rate was 97.76 percent.

These studies used pattern-based approaches. The problems presented in the previous section still exist.

3 The Memory Symptom of Program Execution

A *symptom* is defined as the usage of computer resources when a program is running. Among all the resources, the CPU and the memory are necessary for program execution. In this study, memory symptom is used for virus detection. The memory usage of two programs is shown in Figure 1. One is a normal program (Media Player) and the other is a virus (Netsky). The x-axis represents the time and y-axis represents the memory usage in kilobytes. It is easy to distinguish the two curves.

Two examples are also shown in Figure 2 and Figure 3. In Figure 2, ten curves for ten mutations of the Netsky virus are shown. It can be clearly seen that the ten curves are almost the same.

In Figure 3, eight curves for the memory usage in PowerPoint is shown. Different sizes of PowerPoint files were opened to illustrate the influence of file size on memory usage. The ending "_1" given to the names of the curves

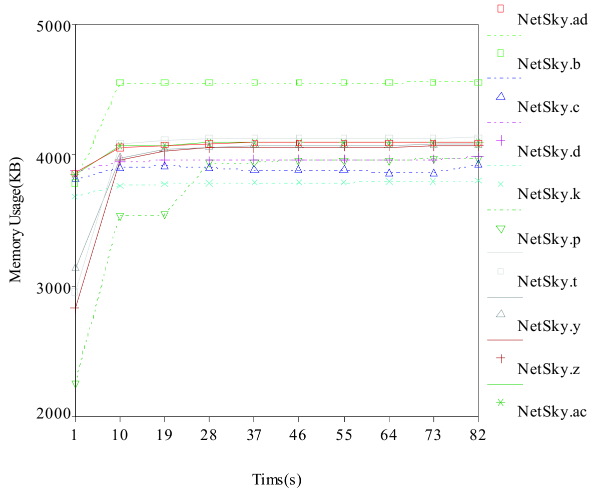


Figure 2: Memory usages of ten Netsky virus mutations

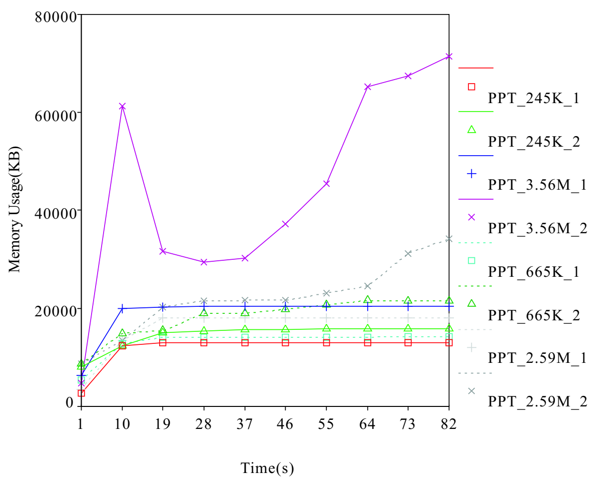


Figure 3: Memory usages of PowerPoint for different file sizes and user operations

indicates that the respective PowerPoint file was not used after it was opened. The ending "_2" given to the names of the curves indicates that the respective file was used after it was opened. These curves are almost the same except for the curve "PPT_3.56M_2". The rapid increase in memory usage is caused by a series of copy-paste operations of picture objects. According to these curves, the file size has only a little effect on memory usage. The file being worked on by a user may affect memory usage more. In general, file-type viruses usually do not have a user interface. A user working on a file is not considered in our research.

Based on the above examples, we can see that the memory symptom represents specific characteristics of programs. It is feasible to detect viruses based on the memory symptom.

4 The Virus Detection Approach

The procedure of the proposed approach is shown in Figure 4. It consists of the following three steps:

1) Sample memory usage: Memory usages are sampled over a fixed interval.

2) Encode: Memory usages are encoded as a series of codes, called curve codes.

3) Match and compute the CF value: The memory symptom of an unknown program is matched with symptoms of sample programs, and a CF value is computed to represent the probability that the unknown program could be a virus.

A set of viruses and normal programs are chosen as samples. They are processed using steps one and two and then their memory symptoms are stored in the sample database. An unknown program is processed using the same steps. Then its memory symptom is matched with those in the sample database, and a CF value is outputted. The above three steps are presented in more detail as follows:

Table 1: The sampled memory usage of the Netsky virus

Time (sec.)	Memory Usage (KB)
1	3,868
4	3,984
7	4,020
10	4,060
...	...

1) Sample memory usage
Memory usage is sampled for a fixed interval and period after the program is executed. Assume the interval is three seconds and the period is 90 seconds. The memory usages of the Netsky virus are listed in Table 1. It is obvious from this table that the memory usages of the Netsky virus stay around four kilobytes after program execution.

2) Encode
The sampled memory usages are encoded in this step. The encoded result, the curve code, is defined to represent the curve of the memory usage. The difference between two successive sampled memory usages is computed. Then the curve code is generated according to the encoding table given in Table 2. There are 39 codes in the encoding table. The 'value' column is defined for the next step. Each code represents a range of memory usages. For example, the range for code 'R' is from 3,000 to 3,999 kilobytes. Therefore, if the difference of two successive sampled usages be-

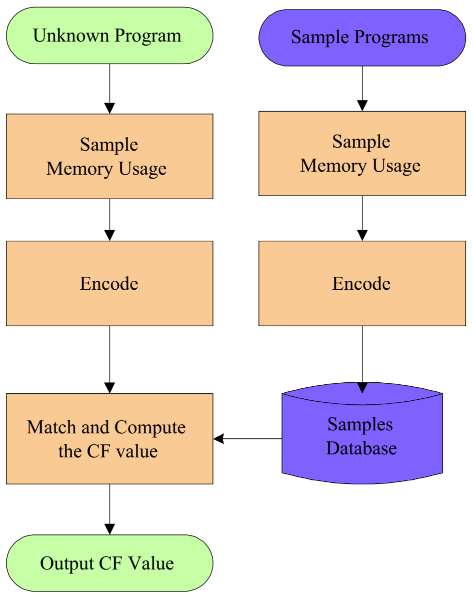


Figure 4: The procedure for the memory symptom-based virus detection approach

longs in this range, code 'R' is generated. Since the difference may be positive or negative, code 'Z' is used when the difference is zero. Codes 'A' to 'S' are used when the difference is positive. Codes 'a' to 's' are used when the difference is negative. The ranges of codes are not equal. They are defined according to our observation of the memory usages of programs. The sampled memory usages shown in Table 1 are encoded into the curve codes listed in Table 3.

Table 3 shows 30 samples for the 90-second sampling period, and 29 curve codes are generated and listed at the bottom of the table.

3) Match and compute the CF value
In this step, a virus is detected by matching the unknown program with the sample programs. There are four measurements for the matching step.

a. *CodeSum*: It measures the sum of the curve codes.

b. *CodeDiffSum*: It measures the sum of the differences of the curve codes.

c. *UsageSum*: It measures the sum of the memory usages.

d. *UsageDiffSum*: It measures the sum of the differences of the memory usages.

The measurements of the curve codes and memory usages can provide information from two different aspects. The results of the four measurements are then computed as the CF value according to the certainty factor model. These four measurements are described in more detail as follows:

Table 2: The encoding table for memory usages

Code	Value	Difference (KB)
S	19	$\geq 4,000$
R	18	$3,000 \sim 3,999$
...	...	...
B	2	$5 \sim 9$
A	1	$1 \sim 4$
Z	0	0
a	-1	$-1 \sim -4$
b	-2	$-5 \sim -9$
...	...	...
r	-18	$-3,000 \sim -3,999$
s	-19	$\leq 4,000$

1) *CodeSum*: The curve code is the difference between two successive memory usages. If the curve codes are summed, the result is usually small. For example, the sum of the curve codes of the Netsky virus in Table 4 is only 25. Therefore, the sum from the first to the ith code is accumulated. The accumulated values of the Netsky curve codes are shown in Table 4. The measurement of *CodeSum* is based on the accumulated values.

Assume there are n curve codes. The ith curve code of the sample program and unknown program is denoted by CS_i and CU_i, respectively. The accumulated values of the sample and unknown programs are denoted as CS_Sum_i and CU_Sum_i, respectively. The equations of CodeSum are shown below.

$$CS\_Sum_i = \sum_{j=1}^{i} CS_J \quad (1)$$

$$CU\_Sum_i = \sum_{j=1}^{i} CU_J \quad (2)$$

$$CodeSum = \frac{\min(\sum_{i=1}^{n} |CS\_Sum_i|, \sum_{i=1}^{n} |CU\_Sum_i|)}{\max(\sum_{i=1}^{n} |CS\_Sum_i|, \sum_{i=1}^{n} |CU\_Sum_i|)} \quad (3)$$

Equations (1) and (2) are used for the computation of the ith accumulated value. In Equation (3), *CodeSum* is defined as the minimum sum of CS_Sum_i or CU_Sum_i divided by the maximum sum. However, CS_Sum_i or CU_Sum_i may be negative. *CodeSum* is based on the absolute value of CS_Sum_i and CU_Sum_i. The result of *CodeSum* is within zero to one. If the result is close to one, this means that the *CodeSums* of the sample and unknown programs are close. Otherwise, the result is close to zero. For example, assume that the values of two curve codes, 'SBb' and 'OAC', are 19, 2, -2 and 16, 1, 3, respectively. Their corresponding

Table 3: The curve codes of the Netsky virus

Sample Time (sec.)	Memory Usage (KB)	Difference	Code
1	3,868		
4	3,984	116	**G**
7	4,020	36	**D**
10	4,060	40	**E**
...	...	...	...
Curve Code = GDEBCAZBAZZZZZZZZZZZZZZZZZZZ			

accumulated values are 19, 21, 19 and 16, 17, 20, respectively. The sums of the accumulated values are 59 and 53, respectively. Therefore, the measurement of $CodeSum$ equals 53/59, or, 0.89.

2) $CodeDiffSum$: This measurement is based on the sum of differences of curve codes. Its equation is shown below.

$$minCurveCode = \text{the minimum value among } CS_i \text{ and } CU_i \text{ for } 1 \leq i \leq n$$
$$CodeDiffSum = 1 - \frac{\sum_{i=1}^{n} |CS_i - CU_i|}{\sum_{i=1}^{n} [\max(CS_i, CU_i) - minCurveCode]} \quad (4)$$

In Equation (4), $minCurveCode$ is defined as the minimum value among all of CS_i and CU_i. The sum of the differences of the curve codes is divided by the sum of the maximum CS_i or CU_i. However, CS_i or CU_i may be negative or positive. Each maximum value is subtracted $minCurveCode$ to ensure that the denominator is larger than the numerator. The division result is within zero to one. Then the number one is subtracted from the above result in order to represent the same meaning of $CodeSum$. That is, if $CodeDiffSum$ is close to one, it means that the curve codes of the sample and unknown programs are similar. Otherwise, the $CodeDiffSum$ is close to zero. For the same curve codes 'SBb' and 'OAC', the values are 19, 2, -2 and 16, 1, 3, respectively. $minCurveCode$ equals -2. $CodeDiffSum$ equals $1 - (3 + 2 + 5)/(21 + 4 + 5)$, i.e., 0.7.

3) $UsageSum$: The measurement is based on the sum of memory usages. Assume there are n memory usages. The ith usage of the sample and unknown program is denoted by US_i and UU_i, respectively. The computation of $UsageSum$ is shown in Equation (5). In this equation, the minimum sum of memory usages is divided by the maximum sum of the memory usages. The result is within zero to one. If the result is close to one, it means that the usage sum of two programs is similar. Otherwise, the result is close to zero. For example, the usages of two programs are 3868, 3984, 4020 and 4000, 3510, 2000, respectively. $UsageSum$ equals $(4000 + 3510 + 2000)/(3868 + 3984 + 4020)$, i.e., 0.8.

Table 4: An example of accumulated values of curve codes

Codes	Value	Accumulated Value
G	7	7
D	4	11
E	5	16
B	2	18
C	3	21
A	1	22
Z	0	22
B	2	24
A	1	25
Z	0	25
...	0	25
Z	0	25

$$UsageSum = \frac{\min(\sum_{i=1}^{n} US_i, \sum_{i=1}^{n} UU_i)}{\max(\sum_{i=1}^{n} US_i, \sum_{i=1}^{n} UU_i)} \quad (5)$$

4) $UsageDiffSum$: This measurement is used to realize the difference of memory usages at every sample time. The computation of $UsageDiffSum$ is shown in Equation (6). The numerator is the sum of usage differences, and the denominator is the sum of maximum memory usages. One minus the division result causes the result to have the same meaning as the other measurements. That is, if the result is close to one, it means that the memory usages of the two programs are similar. For the example with the memory usages 3868, 3984, 4020 and 4000, 3510, 2000, the usage differences are 132, 474, and 2020. $UsageDiffSum$ equals $1 - (132 + 474 + 2000)/(4000 + 3984 + 4020)$, i.e., 0.64.

$$UsageDiffSum = 1 - \frac{\sum_{i=1}^{n} |US_i - UU_i|}{\sum_{i=1}^{n} (US_i, UU_i)} \quad (6)$$

In order to compute the final measurement from the above four measurements, each measurement is assigned a weight instead of the four measurements being simply averaged. Assume the weights are 1, 1, 5, and 5. An

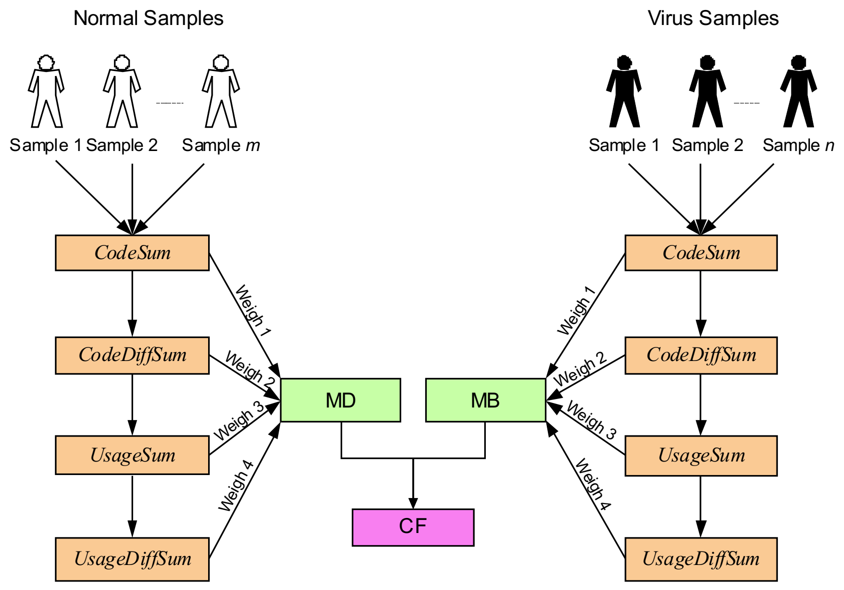


Figure 5: The procedure for matching and computing the CF value

example of computing the final measurement is shown in Table 5. In the table, an unknown program is matched with two samples. The final measurement of Sample-1 is $(0.25 \times 1 + 0.4 \times 1 + 0.28 \times 5 + 0.44 \times 5)/(1 + 1 + 5 + 5)$, i.e., 0.35.

The certainty factor model is then used to represent the possibility that the unknown program is a virus [5]. Sample programs are split into two groups. One is the virus sample, and the other is the normal sample. The final measurement represents the similarity of the unknown program to the sample program. If the sample is a virus, the final measurement is defined as a measure of belief (MB), i.e., the unknown program is believed to be a virus. If the sample is a normal program, the final measurement is defined as a measure of disbelief (MD), i.e., the unknown program is not believed to be a virus. According to the CF-model, the equation for calculating CF from the MB and MD is as follows:

$$CF = \frac{MB - MD}{1 - \min(|MB|, |MD|)} \quad (7)$$

The range of CF is between -1 and 1. If CF is close to -1, it means that the unknown program is normal. If CF is close to 1, it means that the unknown program is a virus. If the unknown program has a CF value between -0.2 and 0.2, the CF value is treated as insignificant. That is, it is too weak to support or deny that the unknown program is a virus.

An example of CF computation is shown in Table 6. There are three normal and three virus samples. The 'MB/MD' column is the average of the final measurements of normal or virus samples. The CF equals 0.86. This means that the unknown program has a higher possibility of being a virus.

Table 5: An example of the computation of the final measurement

Measurement	Weight	Sample-1	Sample-2
CodeSum	1	0.25	0.20
CodeDiffSum	1	0.40	0.34
UsageSum	5	0.28	0.21
UsageDiffSum	5	0.44	0.35
Final Measurement		0.35	0.28

Table 6: An example of CF calculation

Samples	Final Measurement	MD/MB	CF
Normal-1	0.35	MD=0.35	0.86
Normal-2	0.28		
Normal-3	0.41		
Virus-1	0.96	MB=0.91	
Virus-2	0.90		
Virus-3	0.88		

The third step, match and compute the CF value, is shown in the following figure.

5 Experimental Study

5.1 Experimental Metrics

The accuracy of the detection approach is estimated by using the confusion matrix, as shown in Table 7 [2]. The

Table 7: A representation of the confusion matrix

	Virus	Normal
Detected as a Virus	a	d
Detected as Normal	b	e
Unknown	c	f

detection results of the test programs can be classified into six categories. The size of the programs in each category is filled into the corresponding cell in Table 7. In the table, the values 'a' and 'e' represent the number of correct detection. They should be as large as possible. The values 'b' and 'd' represent the number of error detection. They should be as small as possible. The values 'c' and 'f' represent the number of unknown detections, i.e., the CF value is between -0.2 and 0.2. They should be as small as possible.

According to the theory of the confusion matrix, three accuracies are defined as shown below:

1) *True positive rate*: Its equation is $a/(a+b+c)$. This indicates the rate that virus programs are detected as viruses among all the virus programs.

2) *False positive rate*: Its equation is $d/(d+e+f)$. This indicates the rate that normal programs are detected as viruses among all the normal programs.

3) *Unknown rate*: It indicates the rate of unknown programs among all of the programs. It equation is $(c+f)/(a+b+c+d+e+f)$.

These rates are used to verify the proposed approach.

5.2 Experimental Results

A tool for recording memory usages was implemented. A screen shot of the tool is shown in Figure 6. The memory usages of all the processes are listed on the screen and stored into log files. Several parameters were defined in the experiment as follows:

- Sampling period: 90 seconds
- Sampling interval: three seconds
- The size of sample programs: 24 (12 normal and 12 virus)
- The size of test programs: 109 (48 normal and 61 virus)
- Measurement weights: The values were 1, 1, 5, and 5 for $CodeSum$, $CodeDiffSum$, $UsageSum$, and $UsageDiffSum$, respectively.

A variety of normal programs and viruses was used. In the selection of sample programs, the types of memory usages were mainly considered. The memory usages of normal programs were from a few megabytes up to one hundred megabytes. Sample programs were selected from those with small, medium, and large memory usages. The virus samples were mainly selected from those causing serious damage. The selected 24 samples are listed in Table 8.

Table 8: The listing of virus and normal samples

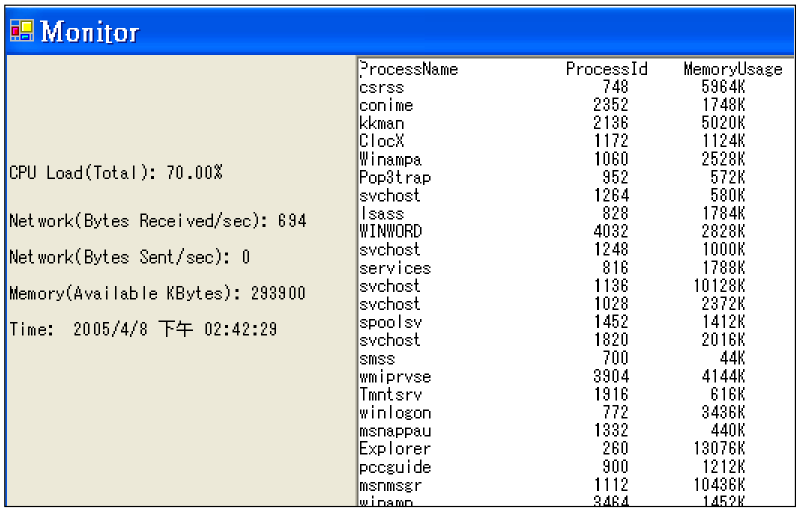
Samples	Name
Normal Samples (12)	PowerPoint
	ACDSee5
	Flash5
	MediaPlayer
	Vb.net
	Word
	Visio
	CuteFTP
	NotePad
	WordPad
	WinRar
	ClockX
Virus Samples (12)	Sobig.f
	Beagle.a
	Sasser.a
	Tanatos.a
	SdBoter.a
	LovGate.n
	Netsky.a
	Deborm.y
	Fizzer
	Bagle.y
	Elkern.c
	Wozer.f

There were a total of 109 test programs. Many popular programs were included in the test programs. In addition, 15 mutations of the Netsky virus were also included in the test programs in order to determine whether all of them can be detected. A partial experimental result of detection using the proposed approach is listed in Table 9.

In Table 9, the symbols '◯', '×', and 'U', denote that detection is correct, wrong, and unknown, respectively. The sizes of the programs were counted and filled into the confusion matrix. The matrix is shown in Table 10.

Based on the above matrix, three rates were computed as follows:

- True positive rate: It equaled $59/(59+0+2)$, i.e., 97 percent.
- False positive rate: It equaled $6/(6+24+18)$, i.e., 13 percent.
- Unknown rate: It equaled $(2+18)/109$, i.e., 18 percent.



```
CPU Load(Total): 70.00%

Network(Bytes Received/sec): 694

Network(Bytes Sent/sec): 0

Memory(Available KBytes): 293900

Time:  2005/4/8 下午 02:42:29
```

```
ProcessName        ProcessId   MemoryUsage
csrss                    748         5964K
conime                  2352         1748K
kkman                   2136         5020K
ClocX                   1172         1124K
Winampa                 1060         2528K
Pop3trap                 952          572K
svchost                 1264          580K
lsass                    828         1784K
WINWORD                 4032         2828K
svchost                 1248         1000K
services                 816         1788K
svchost                 1136        10128K
svchost                 1028         2372K
spoolsv                 1452         1412K
svchost                 1820         2016K
smss                     700           44K
wmiprvse                3904         4144K
Tmntsrv                 1916          616K
winlogon                 772         3436K
msnappau                1332          440K
Explorer                 260        13076K
pccguide                 900         1212K
msnmsgr                 1112        10436K
winamp                  3464         1452K
```

Figure 6: The screen shot of a tool for recording memory usages

Table 9: A partial experimental result of 109 test programs

No.	Type	MD	MB	CF	Result
1	normal	0.55	0.23	-0.42	○
2	normal	0.55	0.44	-0.20	U
3	normal	0.55	0.45	-0.19	U
4	normal	0.60	0.29	-0.44	○
5	normal	0.42	0.70	0.48	×
6	normal	0.53	0.41	-0.21	○
7	normal	0.37	0.57	0.32	×
8	normal	0.56	0.52	-0.08	U
9	normal	0.54	0.24	-0.40	○
10	normal	0.57	0.36	-0.33	○
11	normal	0.57	0.56	-0.03	U
12	normal	0.54	0.54	-0.01	U
13	normal	0.40	0.18	-0.26	○
14	normal	0.56	0.35	-0.33	○
...	...	...	...	...	...
95	virus	0.45	0.76	0.57	○
96	virus	0.42	0.73	0.53	○
97	virus	0.33	0.54	0.31	○
98	virus	0.52	0.64	0.26	○
99	virus	0.48	0.79	0.58	○
100	virus	0.51	0.81	0.61	○
101	virus	0.45	0.79	0.61	○
102	virus	0.45	0.79	0.61	○
103	virus	0.45	0.79	0.61	○
104	virus	0.46	0.79	0.61	○
105	virus	0.47	0.78	0.58	○
106	virus	0.52	0.72	0.43	○
107	virus	0.45	0.74	0.53	○
108	virus	0.45	0.76	0.57	○
109	virus	0.44	0.76	0.57	○

Table 10: The confusion matrix of the experimental result

	Virus	Normal
Detected as a virus	59	6
Detected as normal	0	24
Unknown	2	18

According to the above results, 97 percent of viruses were detected correctly. 13 percent of normal programs were misjudged as viruses. That is not very high. 18 percent of programs could not be determined. Most of them were normal programs. This means that most viruses were detected correctly.

6 Conclusion and Future Works

Currently, anti-virus software relies mainly on the update of virus patterns. However, they still suffer from the problems of damage occurring on zero-day, virus scan inefficiency, and mutation of viruses. In this paper, a memory symptom-based virus detection approach was proposed. Viruses can be detected after it starts executing. The experimental results show that viruses can be detected correctly. In the future, other symptoms, such as network bandwidth usages, can be incorporated into this approach to increase the accuracy of virus detection.

References

[1] *CAIDA Analysis of Code-Red*, http://www.caida.org/analysis/security/code-red/index.xml.

[2] *Confusion Matrix*, http://www2.cs.uregina.ca/ hamilton/courses/831/notes/confusion_matrix/ confusion_matrix.html.

[3] J. S. Lee, J. Hsiang, and P. H. Tsang, "A generic virus detection agent on the Internet," in *Proceedings of the Thirtieth Hawaii International Conference on System Science*, vol. 4, pp. 210–219, Jan. 1997.

[4] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Stanford, and N. Weaver, "Inside the slammer worm," *IEEE Security & Privacy Magazine*, vol. 1, issue 4, pp. 33–39, July-Aug. 2003.

[5] M. Negnevitsky, *Artificial Intelligence - A Guide to Intelligent Systems*, *Addison-Wesley*, pp. 74–80, 2002.

[6] T. Okamoto and Y. Ishida, "A distributed approach to computer virus detection and neutralization by autonomous and heterogeneous agents," in *The Fourth International Symposium on Integration of Heterogeneous Systems*, pp. 328–331, Mar. 1999.

[7] E. Schultz, "Worms and viruses: Are we losing control?," *Science Computers and Security*, vol. 23, no. 3, pp. 179–180, 2004.

[8] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of IEEE Symposium of Security and Privacy*, pp. 38–49, May 14-16, 2001.

[9] J. H. Wang, P. S. Deng, et al., "Virus detection using data mining techniques," in *Proceedings of the 37th IEEE International Carnahan Conference on Security Technology*, pp. 71–76, Oct. 2003.

Hsien Chou Liao is an assistant professor of computer science of information engineering at Chaoyang university of technology (CYUT), Taiwan, ROC. He received his B.S. and Ph.D. degree from National ChiaoTung University, Taiwan, ROC, in 1991 and 1998. Dr. Liao is a senior member of IEEE. His research interests include computer security, pervasive computing, and software engineering.

Yi-Hsiang Hwang received the M.S. in Computer Science and Information Engineering from the Chaoyang University of Technology (CYUT), Taichung, Taiwan, ROC, in 2004. His current research interests are in the area of network and computer security.