PAPER Special Section on Internet Technology and its Architecture for Ambient Information Systems

An Automatic Unpacking Method for Computer Virus Effective in the Virus Filter Based on Paul Graham's Bayesian Theorem

Dengfeng ZHANG[†a)], *Nonmember*, **Naoshi NAKAYA**[†], **Yuuji KOUI**[†], *Members*, *and* **Hitoaki YOSHIDA**[††], *Nonmember*

SUMMARY Recently, the appearance frequency of computer virus variants has increased. Updates to virus information using the normal pattern matching method are increasingly unable to keep up with the speed at which viruses occur, since it takes time to extract the characteristic patterns for each virus. Therefore, a rapid, automatic virus detection algorithm using static code analysis is necessary. However, recent computer viruses are almost always compressed and obfuscated. It is difficult to determine the characteristics of the binary code from the obfuscated computer viruses. Therefore, this paper proposes a method that unpacks compressed computer viruses automatically independent of the compression format. The proposed method unpacks the common compression formats accurately 80% of the time, while unknown compression formats can also be unpacked. The proposed method is effective against unknown viruses by combining it with the existing known virus detection system like Paul Graham's Bayesian Virus Filter etc.

key words: *virus, obfuscate, compression, unpacking, Bayesian virus filter*

1. Introduction

Damage due to computer viruses, which will be referred to simply as viruses, has become a serious problem, since personal computers are widely used in universities, research organizations, and enterprises, as well as at home. According to information provided by the Information-technology Promotion Agency of Japan (IPA) on the occurrence of viruses in Japan, in 2007, there were 166 new types of viruses, which is more than the total number of viruses reported in 2006 (156 types), [1]. Of these 166 virus types, 46 of them were reported for the first time. In addition, the appearance frequency of virus variants has increased steadily*. Due to this increase in the appearance frequency of virus variants, the updates for antivirus software have had difficulties in providing up-to-date patches, since it takes time to determine the characteristic pattern (signature) of a new virus. Because the damage caused by viruses spreads rapidly in a short period of time, it is very important to detect unknown viruses.

To detect unknown viruses, the following techniques are available: static code analysis techniques, which analyze the virus at the assembler level [2]; techniques that analyze the behavior of the virus at the network level by running the virus [3]; and techniques that detect an unknown virus using the naive Bayes learning algorithm [4], [5].

Analyzing a virus statically can detect an unknown virus quickly. Strings, which are the characteristic points in the virus binary files, are used well in static virus analysis. The Strings are the data of the printable characters in the binary file, which can be extracted from a virus binary file, including the Dynamic Link Library (DLL) name and the Application Program Interface (API) that are used by the virus, the words that are used in the body of the mail in a virus that has a mail transmitting function, the path of the file or the registry keys in the virus file, and the ASCII strings that appear at random in the virus file. The characteristic points of the virus can be obtained by extracting the Strings from a virus binary file.

Recently, however, virus files are often compressed using some compression tools. The virus file is compressed in such a way that it decompresses itself into the memory at runtime. Since the virus code is compressed and obfuscated, it is difficult to analyze the virus file statically and extract the characteristic points from a compressed virus binary file [5]. Thus, in order to efficiently detect the unknown virus, it is necessary to automatically unpack it first.

Recently, new compression tools have been rapidly created. Since some compression tools are published as open source by the authors, the deformation of the compression tools is publicly available. Under such conditions, the development of a decompression tool cannot catch up with the development of the compression tool. Thus, an unpack technique that is independent of the compression type is necessary to quickly detect a computer virus.

In this paper, an automatic unpack technique for a compressed computer virus, using the decompression routine of the compression tool itself, is proposed. Concretely, the virus is forced to act in Microsoft Windows, on which the virtual machine environment VMWare has been installed, which controls the compression tool in the debug mode and, then, dumps the memory images of the virus at the end of the decompression routine.

Section 2 will present the compression conditions of the present computer virus, while Sect. 3 will describe the proposed technique. Section 4 will consider a Bayesian antivirus filter, which uses Paul Graham's Bayesian learning

Manuscript received July 28, 2008.
Manuscript revised November 17, 2008.
[†]The authors are with Engineering Graduate Course, Iwate University, Morioka-shi, 020-8551 Japan.
[††]The author is with Super Computing and Information Sciences Center, Iwate University, Morioka-shi, 020-8550 Japan.
a) E-mail: zhang@mn.cis.iwate-u.ac.jp
DOI: 10.1587/transcom.E92.B.1119

*A variant of W32/Sober that first appeared in November 2005 had more than 70 types in only 2 months.

algorithm to detect the unknown virus. Section 5 will give the result of experiments and an evaluation of the proposed technique. Finally, Sect. 6 will summarize the results of this paper.

2. Compression Conditions of Current Computer Viruses

The Microsoft Win32 executable file always follows the Windows PE-format [6]. Figure 1 shows the algorithm followed by the PE-format, where the compression tool compresses and obfuscates each section of the executable file, with a decompression routine at the head of the compressed data. This becomes a new executable file that follows the Windows PE-format. When the compressed executable file is run, the decompression routine is executed first, and it decompresses the compressed data to the memory. Next, the pointer is moved to the Original Entry Point, and the original file is executed.

The size of the compressed virus becomes small and easy to distribute through the network. As well, it becomes difficult to extract the virus characteristic points (Windows API, Strings etc.) from the compressed virus file. For comparison's sake, Windows's Notepad 5.1 was compressed using UPX2.0, and then **Strings** were extracted, and the Windows API was determined using the GNU command Strings, which works on UNIX. Table 1 shows the results. The number of Windows APIs and **Strings** that can be extracted from the compression is less than the number that can be extracted before compression. However, there were only three common Windows APIs between the before-and-after compression files. Thus, the three APIs are the minimum necessary for the compression tool. As mentioned above, it is well known that the characteristic points of the virus cannot be extracted from the compressed virus file.

2.1 The Need for Unpacking

As previously mentioned, the characteristic points of the original virus file are hidden by compression. The paper [5] shows that, using Paul Graham's Bayesian Virus Filter, a compressed nonvirus file will be classified as a virus file, since the extracted features will be the characteristic points of the compressed file and not that of the original uncompressed file.

In order to determine the characteristics of viruses, the data provided by the Information Sciences Center, Iwate

Table 1 Number of successfully extracted features with or without compression.

	before	after	matching rate(%)
Windows API	173	8	1.7%
Strings	208	74	1.5%

Table 2 Virus compression types.

Compression Type	ratio(%)
UPX	36%
Pecompact	12%
ASPack	10%
FSG	10%
PEX	8%
Morphine	6%
MEW11	6%
Yoda's protector	2%
Other	5%
Uncompressed	5%

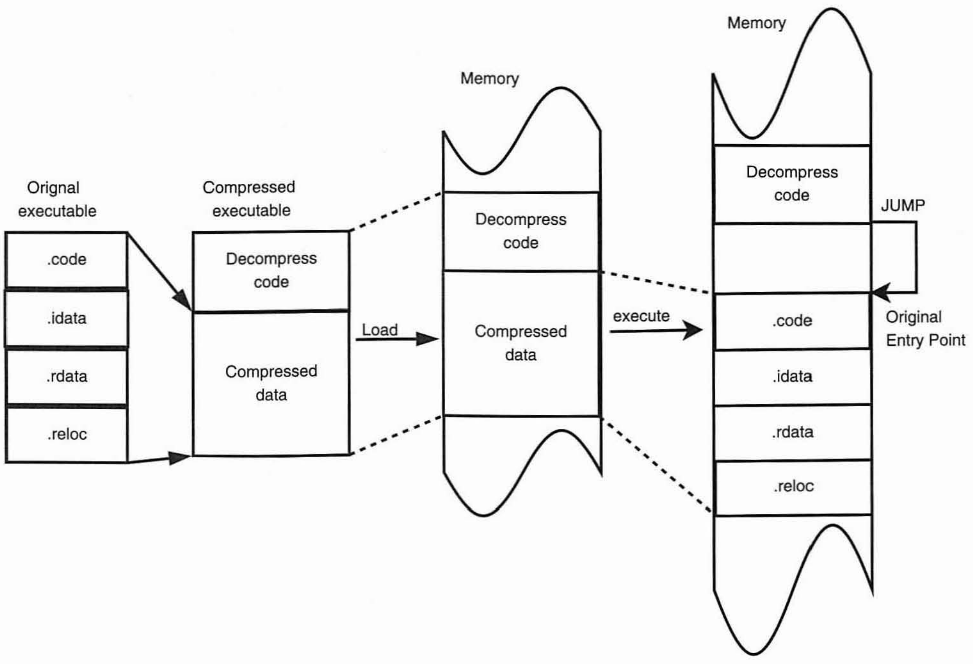


Fig. 1 The memory conditions when the compressed file is executed.

University, in 2005 using the antivirus tool F-Prot [7] were examined. Table 2 shows the results of comparing the different types of compression used for the viruses. The compression types were obtained using the compression type distinction tool PEiD [8].

In addition, it should be noted that over 90% of all viruses were compressed. Thus, it can be concluded that an unpack technique is necessary for virus detection.

3. Unpack Techniques

3.1 Summary

A compressed virus must first be unpacked before it can be analyzed. Currently, various techniques are available for unpacking a virus. For example, a technique that unpacks a virus by using a suitable decompression algorithm for analyzing the compression type of the compressed virus was proposed in [9]. However, this technique only unpacks the viruses that are compressed by well-known compression algorithms such as LZ77 or LZW. As well, this method cannot unpack a virus that has been obfuscated. Another unpack technique, proposed in [10], executes the virus and dumps the main module of the virus process from memory by enumerating and comparing the system process list. However, recent viruses usually create more than one process when they are executed. As well, viruses hide their processes so that the processes cannot be enumerated using Windows's API. For example, when W32/Bagle is run, it displays an alert window, and it does not create any virus processes until the alert window is closed. On the other hand, the W32/Warezove virus first executes the standard Windows application Notepad.exe when it is run, which makes the virus's own process creation timing change dynamically. For such viruses, it is difficult to dump the virus process because the virus process cannot be obtained.

In this paper, an unpack technique using a compression tool's own decompression routine is proposed. When a compressed virus is run, the decompression routine that is created by the compression tool is first executed. Then, the decompression routine decompresses the virus to memory. By the end of the decompression routine, it jumps to the Original Entry Point and the virus begins to run.

Since the proposed technique uses the compression tool's own decompression routine, the compression algorithm of the virus is irrelevant. Of note, the proposed technique works by controlling the compression tool, not the virus. Furthermore, since the proposed technique stops before the virus is executed, virus antidetection is also irrelevant. For example, if the virus monitors the VM machine or the time, virus execution will be stopped. This does not influence the proposed technique, even though the proposed technique works on the VM machine.

In the proposed technique, the virus is executed in Windows Debug mode and is controlled with the one-step mode using Breakpoints. At the end of the decompression routine, the processes are stopped and the decompressed memory image is dumped from memory. The dumped memory image contains all the characteristic points of the original virus file. Figure 2 describes the flow of the proposed technique.

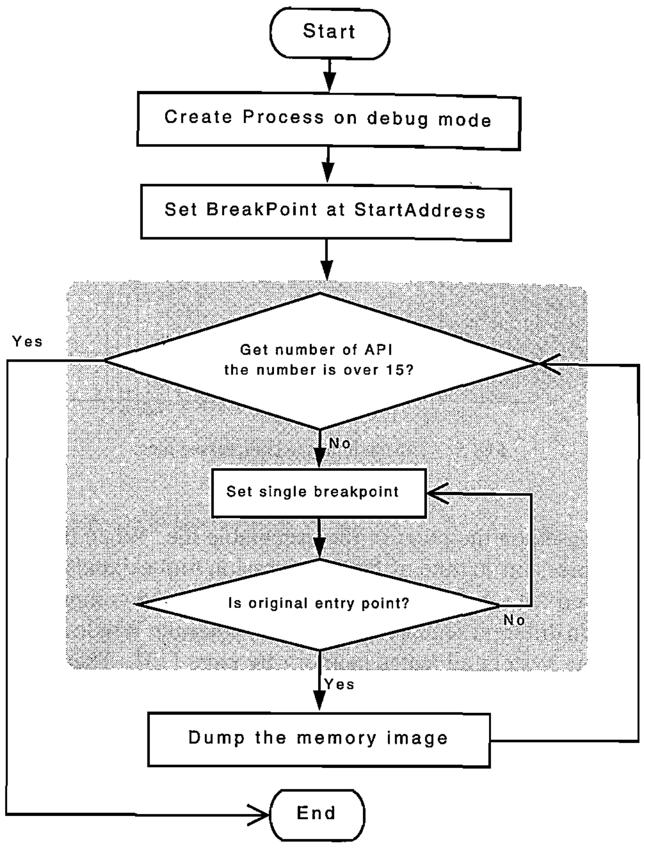

Start
Create Process on debug mode
Set BreakPoint at StartAddress
Get number of API the number is over 15?
Yes
No
Set single breakpoint
Is original entry point?
No
Yes
Dump the memory image
End

Fig. 2 The operation flow of the automatic unpacking system.

At first, the virus process is created in the Windows Debug mode. A breakpoint is then set at the StartAddress, which is the begin address of the process. When the process jumps to the Original Entry Point, this process is stopped, and a memory dump is performed.

Since the address of the Original Entry Point is different in each compression type, it is important to find the Original Entry Point in the automatic unpack technique. However, because some viruses are multiplex compressed, it is insufficient to judge whether the compression is decompressed completely with the Original Entry Point only. A new indicator is necessary to judge whether or not the compressed virus is completely decompressed.

Recently, some compression tools have an antidebug function using the Structured Exception Handling (SEH) [11], [12], which makes it more difficult to debug the compressed virus. A solution to this problem is presented later in this section.

3.2 Determining the Original Entry Point

In the automatic unpacking system, it is important to determine the Original Entry Point to dump the memory im-

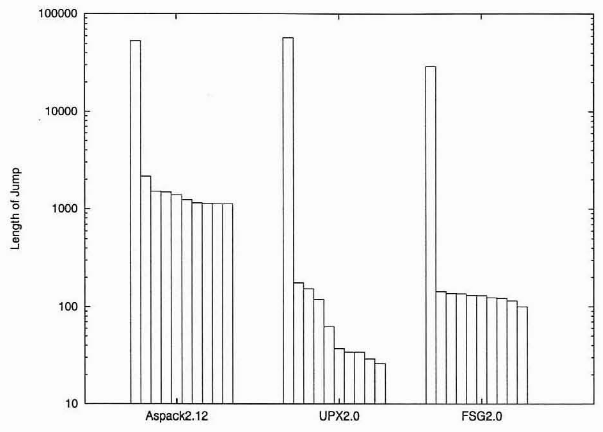


Fig. 3 Top ten jumps in compressed files.

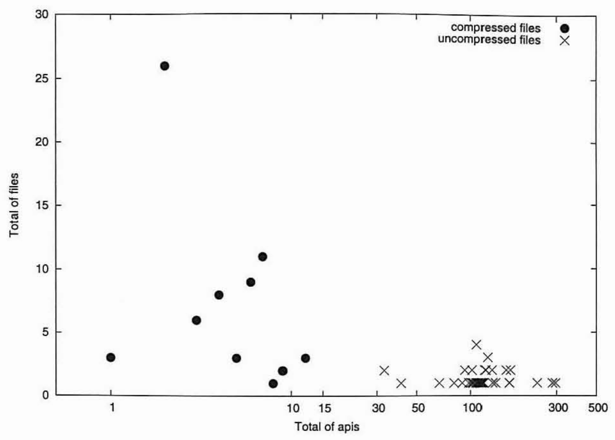


Fig. 4 Number of extracted APIs from compressed and uncompressed files.

age based on the point of the executable file. When the decompression routine of the compression tool expands compressed data into memory, it has to expand the compressed data in different segments from itself, since the Windows operating system allocates memory by page [13]. Therefore, the jump from the decompression routine code to the original entry point is a over segment jump, which will be referred to as a "big jump." The presence of a "big jump" can be used to determine whether or not all of the compressed data has been decompressed.

Notepad Ver.5.1 that was compressed with UPX2.0, Aspack2.12, and FSG2.0 was analyzed using the assembler level analyzing debugger OllyDbg [14]. The top ten jumps in each compressed file are shown in Fig. 3, where the numeric value of the y-axis shows the jump steps. Obviously, a "big jump" occurs in every compressed file when the process jumps to the Original Entry Point. This implies that the compress tool has unpacked the original file data to a different segment from itself. The Original Entry Point can be determined by finding the "big jump" in the virus process.

3.3 Judging the End of the Decompression Routine

To judge whether the compression data are completely decompressed, a test was performed using the following data:

Compressed Virus The viruses that were compressed by some compression tools, such as UPX, ASPack, or FSG. In 2005, there were 53 virus types in the virus database created by the Information Sciences Center, Iwate University.

Uncompressed Virus The viruses that were not compressed. In 2005, there were three uncompressed virus types in the virus database created by the Information Sciences Center, Iwate University.

Compressed Executable File The Win32 executable files that were compressed using the compression tools. In this experiment, Notepad.exe and telnet.exe were compressed using seven different types of compression tools.

Uncompressed Executable File The general Win32 executable files that were not compressed. In this experiment, 40 types of uncompressed executable files were randomly chosen from Windows.

Thus, there was a total of 100 different patterns from which Strings were extracted and the number of Windows API were counted. Figure 4 shows the number of extracted APIs from the compressed and uncompressed files.

As Figure 4 shows, the Windows APIs are barely extracted from the compressed executable files. The Windows APIs that are extracted are the API associated with the decompression routine of the compression tool. By comparison, more than 100 Windows APIs are extracted from the uncompressed executable file. However, there were no compressed executable files with more than 15 APIs. Therefore, it is possible to determine whether an executable file is compressed by counting the number of Windows APIs that can be extracted from it. The threshold number of Windows APIs was set at 15 in the proposed technique.

3.4 SEH Processing

Structured Exception Handling (SEH) is an exception processing service that is provided by the Windows operating system. Usually, exception processing code is only described in the program with _try/_except. An exception management structure and call-back function are formed automatically by the compiler. The exception processing is run when an exception occurs in the program.

Figure 5 is the structure diagram of SEH. When an executable file is run in Windows, the thread information will be saved in the Thread Environment Block (TEB) structure. The address of TEB is always stored in the Segment Register (FS). The SEH structured exception processing is stored as a chain in the memory, where the address of SEH is the first element of the TEB (FS:[0]). In passing, the value of FS:[30h] is the linear address of Process Environment Block (PEB) which will be used in Sect. 3.6.

When an exception occurs in a program, Windows runs the exception processing in the following algorithm:

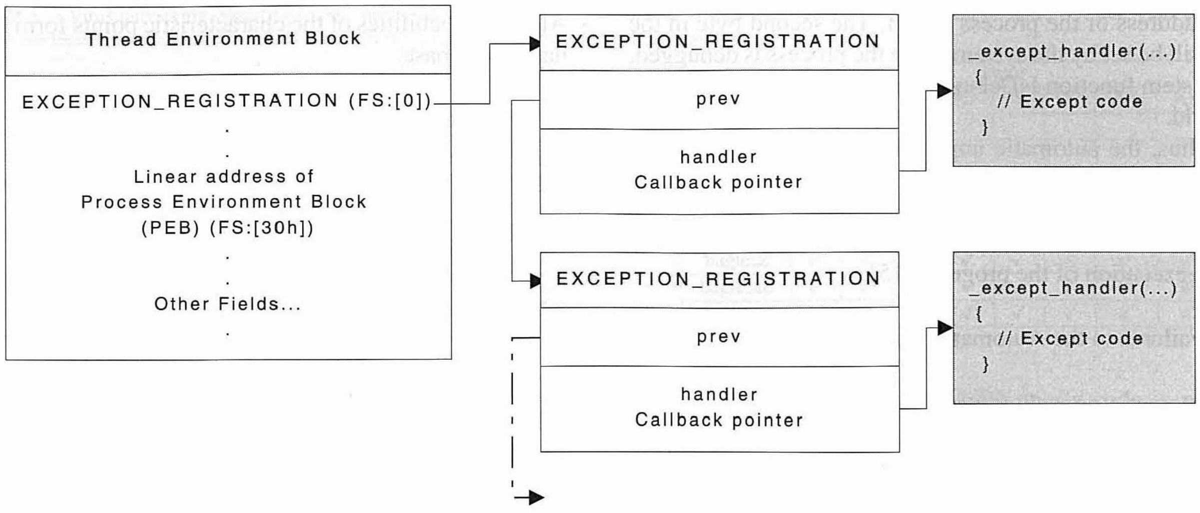

Thread Environment Block
EXCEPTION_REGISTRATION (FS:[0])
Linear address of Process Environment Block (PEB) (FS:[30h])
Other Fields...
EXCEPTION_REGISTRATION
prev
handler Callback pointer
_except_handler(...) { // Except code }
EXCEPTION_REGISTRATION
prev
handler Callback pointer
_except_handler(...) { // Except code }

Fig. 5 Detailed specifications of SEH processing.

1. The OS gets the address of the SEH structure chain from the FS register of the current process (thread). The SEH structure has two members: one is the address of the next SEH structure in the chain, and the other is the address of its exception processing.
2. The OS looks for the suitable exception processing along the SEH structure chain.
3. If suitable exception processing is found, it will be processed.
4. The OS runs the default exception processing if suitable exception processing cannot be found.
5. By the end of exception processing, the code with the exception will be run again.

SEH is used to prevent debugging by many compression tools. The compression tool makes exception interruption occur forcibly by accessing an illegal address intentionally or doing zero division and adding another decompression code or antidebug code in the corresponding SEH exception processing area. When analyzing a virus in debug mode, such an exception cannot be predicted, which makes it difficult to debug the virus.

In the proposed technique, by managing the breakpoints that are set by the unpacking program itself, the exception processing of the debugged program is run when an exception occurs that is not specified. Therefore, the unpacking program will not be stopped at an unexpected breakpoint.

Since the detailed specifications of SEH processing, such as the convolution operation of the stack when an exception interruption handle is started in chains, have not been made publicly available by Microsoft, new techniques that abuse SEH keep appearing. Thus, measures against compression tools that have this antidebugging function using SEH will be tackled in the future.

3.5 System Call Processing

The Win32 executable file works by calling the Windows API (System Call). The Windows API library (DLL file) will be loaded into the system block in memory, and the program code will be loaded into the user block in memory.

In general, a program calls Windows API functions using a CALL command, but the compression tool scans the memory or analyzes the Export Table of the DLL file to get the address of an API function, and it calls an API function using a JUMP command to hide the calling Windows API function action. Thus, it is impossible to determine the program's system call by statically analyzing the program code.

The automatic unpacking program spends time in the one-step mode during the system call, because it is unsafe to set a breakpoint in system memory. The return address is obtained from the system's stack when the compressed program runs in the system's memory block, and the unpacking program sets a breakpoint at the return address that is in the user memory block. Therefore, to determine a system call, it is not necessary to analyze the program code; all that is required to set a breakpoint in the user memory block.

3.6 Antidebugging

Recently, antidebugging technology has been implemented by many compression tools. The effective and commonly used antidebug technique is as follows:

1. Check the process of some debug tools that are commonly used, such as Ollydbg or SoftICE.
2. Use the IsDebuggerPresent function of the KERNEL32.DLL library, or check the value of the second byte in the Process Environment Block (PEB)

The former technique does not correspond to the automatic unpacking program. As mentioned in Sect. 3.4, when an executable file is run in Windows, the value of FS:[30h] refers to the address of the process's PEB. The second byte in the PEB will be set by the system when the process is debugged. The system function IsDebuggerPresent works by checking this field.

Thus, the automatic unpacking program fools antidebuggers that use the IsDebugger Present function by setting the second byte in the PEB to zero during the creation of a thread. This field can be reset to zero without consequences for the execution of the program [15].

3.7 Failures in the Automatic Unpacking Program

The automatic unpacking program enumerates the process list of the system before loading a virus file to get the virus process handle by comparing the processes before and after the virus is executed. When the automatic unpacking program fails, such as failure on setting a breakpoint, the main module of the virus process will be dumped from the memory.

4. Paul Graham's Bayesian Virus Filter

The Paul Graham's Bayesian Virus Filter [5] detects an unknown virus by using Paul Graham's Bayesian theorem. Paul Graham's Bayesian theorem was originally developed for a spam mail filter [16], but it has been adapted to the detection of unknown viruses by changing some of the parameters. It contains a learning module and a detection module.

4.1 Learning Module

The learning module learns the well-known virus in the following manner:

1. Determine whether a given file is a virus file using antivirus software.
2. Extract the **Strings** that are the characteristic points from the virus files and the no virus files.
3. Calculate the probability $P(s)$ that a certain characteristic point s is a component of a virus.

The total number of virus files is n_{bad}, the total number of no virus files is n_{good}, the number of times that a certain characteristic point s appeared in the virus files is b, the number of times that a certain characteristic point s appeared in the no virus files is g, and the probability $P(s)$ is calculated as follows:

$$P(s) = \frac{b/n_{bad}}{2g/n_{good} + b/n_{bad}} \tag{1}$$

Equation (1) is the original equation [16] used to calculate the probability of each characteristic point. In Eq. (1), all the parameters with subscript "good" are doubled to bias the probabilities slightly to avoid false positives. The probability $P(s)$ of a characteristic point that only appears in the virus file is initialized at 0.99, while it is initialized to 0.01 for a characteristic point that only appears in a no virus file. All the probabilities of the characteristic points form the signature database.

4.2 Detection Module

The detection module detects an unknown virus as follows:

1. Extract the characteristic points from a target executable file.
2. Get the virus probability of each characteristic point from the signature database. Initialize the default virus probability as 0.4 if a characteristics point does not exist in the signature database.
3. The largest 15 virus probabilities form the base probability P_{base}. Based on the virus probabilities, the target executable file's characteristic points are chosen to calculate the target executable file's virus probability P_v. The target executable file will be judged to be a virus if P_v is greater than 0.9.
   In this paper, P_{base} is initialized to 0.499.

The virus probabilities that are chosen are defined as P_i, and the virus probability of the target executable file P_v is calculated using the following formula:

$$P_v = \frac{\prod_{i=1}^{15} P_i}{\prod_{i=1}^{15} P_i + \prod_{i=1}^{15}(1 - P_i)} \tag{2}$$

5. Experiment and Evaluation

5.1 The Experimental Environment

The automatic unpacking technique proposed in this paper was tested on a personal computer running WindowsXP Professional SP2 with an isolated VMWare 5.0 system. Since the experiment is run on an isolated computer, virus spreading is not an issue. Furthermore, since the experiment is run on VMWare, it is easy to recover the Windows system even if it is infected by a virus. Thus, it can be assumed that all of the experiments are run in the same environment.

5.2 The Experimental Data

In the experiment, all of the 56 virus types that were marked by the Information Sciences Center, Iwate University, in 2005 were used as the experimental data. Of these types, 42 types were compressed using a compression tool, such as UPX, ASPack, FSG, or Morphine, eleven types were compressed by unknown compression tools and the remaining three types were not compressed.

5.3 The Experimental Results and Evaluation

Table 3 shows the results of the experiment. Of the compressed viruses, 83.9%, or 47 types, were successfully unpacked.

Table 3 Virus unpacking conditions in the proposed technique.

			Number of APIs	
Virus Name	Compress type	time	before	after
Bagle.J	UPX 0.89	1m26s	6	61
Bagle.K	UPX 0.89	1m31s	7	55
Bagle.M	UPX 0.89	1m32s	7	86
Bagle.N	UPX 0.89	1m45s	7	86
Bagle.Y	UPX 0.89	1m53s	7	81
Bagle.Z	UPX 0.89	1m23s	7	81
Bagle.AB	UPX 0.89	1m21s	7	81
Bagle.AF	UPX 0.89	1m25s	8	91
Bagle AI,AM,AP,AQ	PEX	-	7	15
Bagle BA,BC,BK	unknown	-	6	15
Dumaru.A	UPX 0.89	0m44s	4	50
Buchon.B	UPX 0.89	14m02s	4	32
Buchon.F	UPX 0.89	7m27s	4	19
Bugbear.B	UPX 0.89	9m25s	6	95
Klez.H	uncompressed	-	-	-
Erkez.B	FSG 1.33	0m59s	2	38
Erkez.D	FSG 2.0	0m53s	3	38
Explet.A	FSG 1.33	1m10s	6	24
Lovgate.R	ASPack 2.12	27m53s	7	24
Lovgate.X	ASPack 2.12	32m21s	12	24
Lovgate.Y	ASPack 2.12	33m16s	8	24
Lovgate.AC	ASPack 2.12	29m03s	12	24
Lovgate.AD	ASPack 2.12	29m41s	12	24
Maslan.C	uncompressed	-	-	-
Mydoom.L	upx 0.89	1m28s	5	70
Mydoom.M	upx 0.89	1m36s	5	70
Mydoom.Q	upx 0.89	2m12s	6	101
Mydoom.BO	Morphine 1.4-1.7	3m45s	2	77
Mydoom.BQ	Morphine 1.4-1.7	7m11s	2	77
Mydoom.BT	PESpin 0.3x-0.4x	x	x	x
Mytob.B	FSG 1.33	3m19s	2	112
Mytob.C	yoda's Protector 1.3	-	2	116
Mytob.M	Morphine 1.4-2.7	x	x	x
Mytob.V	unknown	38m55s	2	224
Mytob.AG	unknown	36m31s	2	113
Mytob.AH	MEW11 1.2	45m51s	2	115
Mytob.AP	unknown	34m51s	2	114
Mytob.AS	unknown	2m53s	2	124
Mytob.BB	unknown	34m53s	2	114
Mytob.BM	unknown	34m51s	2	114
Mytob.BV	MEW11 1.2	26m33s	2	84
Mytob.CH	MEW11 1.2	31m40s	2	77
Mytob.DH	PECompact 2.x	0m17s	4	116
Netsky N,X	tElock 0.96	x	x	x
Netsky P,Q	FSG 1.0	x	x	x
Netsky.S	unknow	1m55s	6	73
Netsky.T	unknow	1m57s	6	73
Netsky.W	UPX 0.96	3m84s	5	74
Swen.A	uncompressed	-	-	-

The number of APIs that are extracted from the unpacked virus file that had been compressed using compression tools, such as UPX, ASPack, FSG1.33, FSG2.0, and MEW11, increases from the original virus file. As well, in viruses that are compressed using Morphine, some can be unpacked, while others cannot be unpacked, because, since the source code for Morphine is publicly available, the virus authors can modify the code themselves.

A similar problem occurs for the compression tool UPX. For most viruses, it takes only 1 minute to unpack such a virus, while for the Buchon series viruses, it takes on average 10 minutes to unpack the viruses. This can be attributed to the fact that the UPX tool's code is publicly available.

In the experiment, seven of the 11 unknown compression type viruses were unpacked. Thus, it can be said that the proposed technique is effective in identifying unknown compressed viruses.

Table 4 Bagle series virus detection conditions using Paul Graham's Bayesian Filter (original).

	input							
signature	J	K	N	O	Y	Z	AB	AE
bagle_j	✓	✓	✓	✓				
bagle_k	✓	✓	✓	✓				
bagle_n	✓	✓	✓	✓	✓	✓	✓	✓
bagle_o	✓	✓	✓	✓	✓	✓	✓	✓
bagle_y	✓	✓	✓	✓	✓	✓	✓	✓
bagle_z	✓	✓	✓	✓	✓	✓	✓	✓
bagle_ab	✓	✓	✓	✓	✓	✓	✓	✓
bagle_ae	✓	✓	✓	✓	✓	✓	✓	✓

5.4 The Experimental Evaluation Using the Bayesian Virus Filter

The Bagle virus series that was unpacked in the above experiments was tested using Paul Graham's Bayesian Virus Filter to determine the practical performance of the proposed technique.

Table 4 shows the original Bagle virus series detection conditions as given by [5]. These results will be referred to as the original detection conditions.

Two hundred virus files of the specific variants were entered to each signature. A ✓is used to show that the variants were properly detected. The detection conditions of the Bagle virus series, which was unpacked by the proposed technique, is shown in Table 5. It is obvious that the current results match the original detection rates. In the original study, only viruses that could be unpacked were detected by the decompression tools. However, using the automatic unpacking technique, more variants of the viruses were detected.

6. Conclusions

In this paper, a new unpacking technique that can be used to overcome the recent explosion of compressed viruses is proposed. This technique uses the compression tools' own decompression routine running in Windows debug mode. The original virus's characteristic points can be extracted using the proposed technique, making it possible to detect the unknown virus in real time by combining the unpacking technique with current learning-type virus filters and avoiding the need to spend time determining the original virus's signature.

However, many compression tools have public source code that can be modified by the virus author to defeat or interfere with the above mechanisms. Future work will therefore focus on decreasing the unpacking time and providing more advanced functions.

Table 5 Bagle series virus detection conditions using Paul Graham's Bayesian Filter (unpacked by the proposed technique).

	input																	
signature	j	k	m	n	y	z	ab	ae	af	ai	am	ap	aq	ba	bc	bk	by	nonvirus
bagle_j	✓	✓	✓	✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_k	✓	✓	✓	✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_m	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_n	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_z	✓	✓	✓	✓	✓	✓	✓	✓	✓									✓
bagle_ab	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_ae	✓	✓	✓		✓	✓	✓	✓	✓									✓
bagle_af	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_ai	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_am	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_ap	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_aq	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_ba	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_bc	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_bk	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bagle_by	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

References

[1] IPA, "The computer virus report conditions of 2007," report, Information-technology Promotion Agency Japan, 1 2006.

[2] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," 12th USENIX Security Symposium, p.169, 2003.

[3] N. Nakaya, R. Koike, Y. Koui, and H. Yoshida, "The network system defended from infection of unknown e-mail viruses (network security) (special issue computer security and privacy protection)," Trans. IPSJ, vol.45, no.8, pp.1908–1920, 2004.

[4] M.G. Schultz, E. Eskin, E. Zadok, and S.J. Stolfo, "Data mining methods for detection of new malicious executables," Security and Privacy, pp.38–49, IEEE, Oakland, CA, USA, May 2001.

[5] R. Koike, N. Nakaya, Y. Hagihara, Y. Koui, H. Takakuba, and H. Yoshida, "The unknown viruses detection method using bayes learning algorithm," Trans. IPSJ, vol.46, no.8, pp.1984–1996, 2005.

[6] Microsoft Corporation, "Visual studio,microsoft portable executable and common object file format specification," Windows Hardware Developer Central, 2006.

[7] Frisk Software International, "F-prot." http://www.f-prot.com

[8] PEid. http://peid.has.it/

[9] M. Vnuk and P. navrat, "Decompression of run-time compressed pe-files," Studies in Informatics and Control J., vol.15, no.2, pp.169–180, 2006.

[10] A. Mori, T. Sawada, T. Izhumida, and T. Inoue, "Detection unknown computer viruses — A new approach," Review of the National Institute of Information and Communications Technology, vol.51, no.1/2, pp.73–88, March 2005.

[11] M. Pietrek, "A crash course on the depths of win32 structured exception handling," Microsoft System J., vol.12, no.1, http://www.microsoft.com/msj/0197/Exception/Exception.aspx, 1977.

[12] J. Richter, Programming Applications for Microsoft Windows Fourth Edition, Microsoft Press, 1999.

[13] M.E. Russinovich and D.A. Solomon, "Microsoft windows internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000 (pro-developer)," Microsoft Pr, 2005.

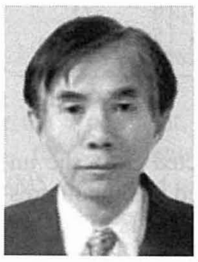
[14] OllyDbg. http://www.ollydbg.de/

[15] N. Falliere, "Windows anti-debug reference." http://www. security-focus.com/infocus/1893, 2007.

[16] G. Paul, "A plan for spam." http://www.paulgraham.com /spam.html, 2002.

Dengfeng Zhang received his B.Eng. and M.Eng. degrees from Iwate University in 2005 and 2007, where he is currently registered as doctoral student since 2007 to the present. He is currently researching network systems and network security.

Naoshi Nakaya received his B.Eng., M.Eng. and Ph.D. degrees from Saitama University in 1994, 1996 and 1999, respectively. He is a research associate in the Department of Computer and Information Sciences at Iwate University and researching network systems, network security and evolutionary algorithms.



Yuuji Koui graduated from the Science University of Tokyo. He joined Mitsubishi Electric Corporation in 1970, and engaged in the design and development of communication systems. He work at Audio-Visual Information Technology Department in Information Technology R&D Center of Mitsubishi Electric Corporation. He received the Ph.D. degree in telecommunication engineering from Tohoku University in 1998. He is a professor in the Faculty of Engineering and Graduate School of Engineering Iwate University and is researching of network systems and network security. He is a member of IEEE and the Information Processing Society of Japan.

Hitoaki Yoshida graduated from Tohoku University. He received his doctoral degree in science from Tohoku University in 1987. He worked at Department of Chemistry in University of Tsukuba from 1987 to 1991. He is an associate professor of Iwate University and working on the research of Complex System, Online Communication and Information Security. He is a member of the Information Processing Society of Japan, the Society of Instrument and Control Engineers, and Council for Improvement of Education through Computers.