PAPER *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems*

A Distributed Approach against Computer Viruses Inspired by the Immune System*

Takeshi OKAMOTO[†], ***Nonmember and*** **Yoshiteru ISHIDA**[†], ***Member***

SUMMARY More than forty thousands computer viruses have appeared so far since the first virus. Six computer viruses on average appear every day. Enormous expansion of the computer network opened a thread of explosive spread of computer viruses. In this paper, we propose a distributed approach against computer virus using the computer network that allows distributed and agent-based approach. Our system is composed of an *immunity-based system* similar to the biological immune system and *recovery system* similar to the recovery mechanism by cell division. The *immunity-based system* recognizes "non-self" (which includes computer viruses) using the "self" information. The *immunity-based system* uses agents similar to an antibody, a natural killer cell and a helper T-cell. *The recover system* uses a *copy agent* which sends an uninfected copy to infected computer on LAN, or receives from uninfected computer on LAN. We implemented a prototype with JAVA^{TM} known as a multi-platform language. In experiments, we confirmed that the proposed system works against some of existing computer viruses that can infect programs for MS–DOS^{TM}.

key words: *computer virus, immune system, agent, self-reference, cell division*

1. Introduction

The explosive expansion of the Internet provides computer viruses with their route of infection to the enormous amount of computers connected to the Internet [1], [2]. Indeed, computer viruses expanded rapidly since 1987. It is said that the strains of computer viruses exceed forty thousands, and six strains on average appeared everyday [3]. To make matters worse, this trend will accelerate due to the development of the Internet and the forum for constructing viruses on the Internet.

After the debut of the computer virus, many anti-virus systems which prevent viruses have been developed. However, since most of them use information specific to virus for detection and repair, they cannot deal with unknown viruses. Some anti-virus systems, indeed, find unknown viruses by a heuristic method [3]–[5]. Unfortunately, it cannot detect all of them and it may even make false alarm. Further, it usually cannot repair the infected programs.

Recently, Kephart proposed the digital immune system which shortens the time from detection of a virus to distribution of a prescription [6]. The technique uses a network to automate the analysis of a virus and all other processes from virus detection to acquisition of the prescription required for the measure against a virus. However, it is a centralized approach where a central computer analyzes the virus and distributes the prescription. Consequently, once the computer is attacked or the path to/from the computer is shutdown, the "immune system" will be completely broken down. Moreover, since the automatic analysis of a virus is done by the "heuristics method," it cannot deal with new viral strains infecting a new file type.

Forrest devised a new detection method that learned from the self/non-self discrimination mechanism of the immune system [7]. The system generates the code group (non-self) which does not exist in the computer system (self). This technique may find a malicious program besides viruses. However, since it is impossible to maintain all the signature of "non-self," the viral codes may be recognized as "self."

Pu proposed a diversification method of the implementation of OS to escape from the virus attack [8]. This diversification strategy seems to be similar to that of biological systems. The strategy is indeed taken not only by the immune system but also by the sex system.

In this paper, we propose a distributed approach against computer virus using the computer network that allows distributed and agent-based approach. Our system is designed based on the mechanism of the immune system of a vertebrate. The immune system is a sophisticated defense system of multi-cellular organisms. Since the change from stand-alone computers to computers connected by LAN is somewhat similar to that from uni-cellular organisms to multi-cellular organisms, some strategies of the immune system may be used in the defense system for networked computers. We have been using the analogy of the immune system as a basis of autonomous decentralized systems [9]. The strategies of "the immune system" used in this paper are as follows:

- It is a distributed system attained by autonomous and heterogeneous agents.
- It uses the information of "host" side (called the "self" information) rather than that of "parasite"

Manuscript received October 8, 1999.

[†]The authors are with the Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441-8580 Japan.

*This paper was presented in part at the 4th International Symposium on Artificial Life and Robotics, 1999, and the 4th International Symposium on Autonomous Decentralized Systems, 1999.

side (called the "non-self" information).

Our system consists of the *immunity-based system* which detects a virus and the *recovery system* which recovers the infected programs. The *immunity-based system* checks the programs which have a danger of being infected by a virus. Checksums and the first few bytes of the programs, etc. are kept as the "self" information in a database. Our system detects virus infection based on the "self" information.

The *immunity-based system* is managed by the *control agent* who plays a similar role to the helper T-cell of the immune system. The *control agent* monitors execution of all programs. When a program is executed, the *control agent* compares the checksum of the "self" information of the program with the checksum calculated just before execution. If the *control agent* detects virus infection, it activates an *antibody agent* and a *killer agent*. The *antibody agent* neutralizes the effect of virus by overwriting the first few bytes of the "self" information. As a result, the virus will no longer be executed. After the neutralization by the *antibody agent*, the *killer agent* removes infected programs. The *killer agent* works on its own in parallel with other agents.

The *recovery system* recovers the infected programs removed by the *immunity-based system*. The *recovery system* is activated, when reported virus infection from an infected computer, or when the *killer agent* detects a virus. A *copy agent* sends/receives an uninfected copy to/from computers on LAN.

Since algorithms of computer viruses do not depend on a specific OS or on a specific machine, computer virus may appear at all platforms in the future. We used a multi-platform language JAVA^{TM} for implementing a prototype.

This paper is organized as follows. Section 2 gives overviews of computer viruses and anti-virus systems. Section 3 presents our approach in detail. Section 4 presents the experimental results of our system with some existing viruses. Section 5 discusses the problems of implementing our system.

2. Computer Viruses and Anti-Virus Systems: State of The Art

2.1 The Overview of Computer Viruses

The original concept of computer viruses may be traced back Von Neumann's works on *Self-reproducing automata* in 1940. The word "computer virus" may be first found in 1987 [10]. In the paper, the computer virus is defined to be a program that can infect by altering other programs. The computer virus discussed in this paper follows this definition. We do not deal with a logic bomb, Trojan horse, computer bacteria, and a worm in this paper.

There are three types of programs.

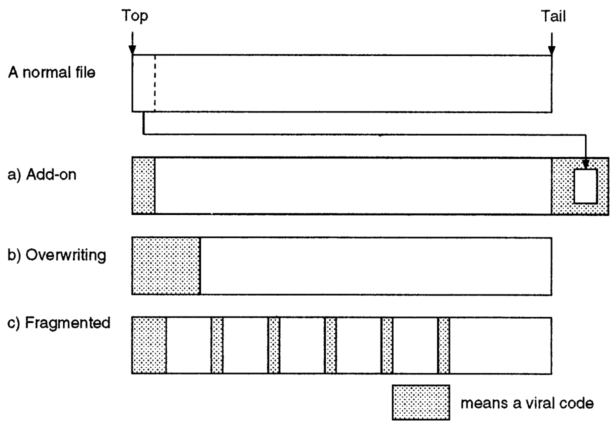


Fig. 1 Types of virus infection.

1. Boot program which exists in a boot sector.
2. Application program and executable system file (under the "*windows*" directory) with extensions .EXE, .COM, .SYS, .DRV, .BIN, etc.
3. Macro program contained in documents and spreadsheet files.

When a virus infects these programs, it alters the entry point of a host program in order to execute its own code before the original host program is executed. Our approach detects a virus by checking the alteration of a file. Documents and spreadsheet files are more often updated compared with programs. Hence, checking those files causes many "false positives"[†] than that for programs. In order to reduce false positive, we will limit the scope of scan only to the executable system files whose alteration is not permitted. The target for our system is viruses that infect executable system files under "*windows*" directory (i.e. second type above). These viruses can be classified into the following three types (Fig. 1) based on the file infection mechanism.

- Add-on infection type [11] (Fig. 1(a)) which attaches main viral codes at the end of a host file.
- Overwriting infection type (Fig. 1(b)) which overwrites viral codes at the head of a host file.
- Fragmented infection type (Fig. 1(c)) which distributes main viral codes inside a host file.

The add-on infection type is the most popular among viruses infecting the files for MS–DOS^{TM}. They attach their own codes to the end of a host file, and alter the first few bytes of the host file in order to execute their own codes before the codes of the host file. After the virus executed all viral codes, they transfer control to the host file (the first few bytes of the original host file at the end of the infected host file) as if the virus

[†]"False positive" means alerting user to virus infection even if there is no virus. It is also called "false alarm." To the contrary, "false negative" means failing to alert user to virus infection even if there is a virus.

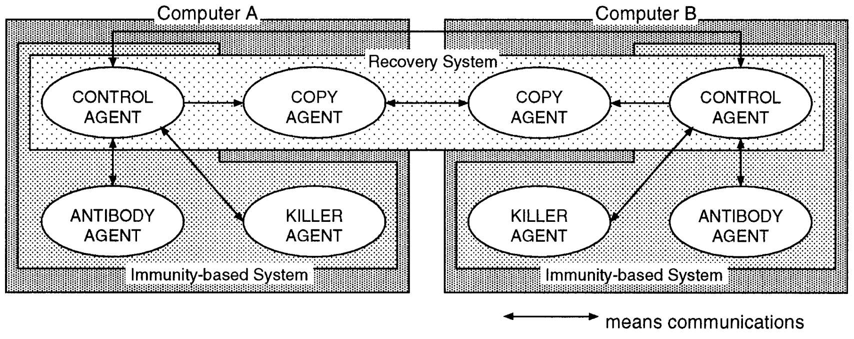


Fig. 2 The overview of our system.

would not exist at all. As a result, they succeed to conceal their own execution from users. Fortunately, this first few bytes added at the end can give commercial anti-virus systems a means to repair the infected files, because the viral codes include the information of transferring control to the host file.

The overwriting infection type can be easily found, because the original host file can no longer function. They resemble "Trojan horse" about changing the function from the original one. Unfortunately, commercial anti-virus systems cannot repair the infected files because the first few bytes of the host files are not kept.

The fragmented infection type is rare, because it needs sophisticated technique of programming them. However, they have a possibility of spreading over the world (e.g. `CIH` virus), because they are capable of escaping from commercial anti-virus systems.

2.2 The Overview of Anti-Virus Systems

Anti-virus systems have been extensively developed against computer viruses. The importance of these systems increases, since the number of computers connected to LAN increases rapidly (compared with workstations whose OS are mostly UNIX). The techniques are roughly divided into the following three types.

- The "scanning method" scans files, boot records and memory to find the specific patterns (called "signature") of viruses.
- The "heuristics method" monitors execution of programs to find the virus-like "behavior" (opening of a program, termination of its own program, etc.).
- The "checksum method" investigates the alteration of the file.

The "scanning method" can detect a virus most correctly out of the above three techniques. It also allows to restore infected files. However, it can detect only the known viruses found in the past. To detect new viruses, the signature must be updated whenever new viruses arise. Moreover, since the detected viruses are analyzed manually, it requires time from first detection to the distribution of the prescription. It would become difficult to catch up the rapid spreading of viruses in recent years.

The "heuristics method" can detect unknown viruses, however, it cannot identify a virus, and restoration of the infection file is almost impossible. Moreover, the viruses detectable by the "heuristics method" are only those infecting a specific file type.

The "checksum method" has a demerit of causing many false positives, since alteration occurs not necessarily at an invasion by a virus (e.g. alteration occurs in a legitimate update of a program.), although all the viruses that alter a file can be detected.

Most of commercial anti-virus systems use both the "scanning method" and the "heuristics method," because their design policy is to avoid a false positive rather than a false negative [5]. Although they are effective in the popular viral strains, they are weak with the new viral strains infecting a new file type (When a new virus which infects JAVA^{TM} files arose, the commercial anti-virus systems could not detect it at that time).

3. The Anti-Virus System

Our system roughly consists of the *immunity-based system* and the *recovery system* (Fig. 2). Our goal is not to pursue the parallelism between the immune system and our system, but to solve the problem of computer viruses. Our system is attained by a distributed and redundant approach, and is different from other anti-virus systems in the following three points.

- Virus detection is done by matching the "self" information (see Sect. 3.1) with the current host programs.
- Neutralization of viruses is done by overwriting the "self" information on the infected files.
- File recovery is attained by copying the same file from other uninfected computer on LAN. The same files potentially under attack (executable system files of OS in our system) will be used as backup with each other through the network.

The remainder of this section describes how these functions are implemented.

3.1 The "Self" Information

Similarly to the recognition of the immune system[†], we propose to recognize "non-self" by comparing it with "self" on the computer system. First, we must define the "self" of the computer system in order to recognize the "non-self." We define the executable system files which originally must not be altered as "self." The reason why executable system files are chosen as the "self" is discussed in Sect. 5. The "non-self" in our system is hence the altered executable system files.

In order to detect the alteration, all information of the "self" must always be kept in the computer system. Although it is an ideal to keep all of the information before infection, it is difficult to keep it in the limited capacity. Our system keeps the information such as the file attributes (a file name, the full path of a file, a file checksum, the size of a file), and a temporary data for restoration (first few bytes of a file) as "self." The file attributes are used for detection of non-self (virus), and the temporary data for restoration is used to overwrite the viral code at the head of the file. Hereafter, these data (the file attributes and the temporary data) will be called the "self" information. All computers on LAN keep their own database of the "self" information in the path specified by an administrator. We suppose that no virus knows the path.

3.2 Immunity-Based System

We propose the *immunity-based system* which neutralizes and removes viruses. The neutralization is done by an *antibody agent* similar to the antibody[††]. The removal is done by a *killer agent* similar to a natural killer cell and a killer T-cell[†††]. The *antibody agent* is activated by a *control agent* similar to a helper T-cell[††††], just before a program of "self" is executed. The *killer agent* periodically run. It is also activated by the *control agent* when the *control agent* receives a report of the infection from other *control agents* in LAN.

The *antibody agent* and the *killer agent* compare the size, the checksum and the path to the current file with those in the "self" information (see Sect. 3.1), and if they differ, it will be considered an infected file. This detection method (the "checksum method") could cause many false positives, however, some remedies are shown in the paper [12]. Our system solves this problem by limiting the target of detection to the executable system files not allowed alteration (see Sect. 5.1 in detail).

If the *antibody agent* detects infection, it overwrites the head of the infected file with the temporary data for restoration saved in the "self" information. In this way, the viral code is neutralized because the virus loses

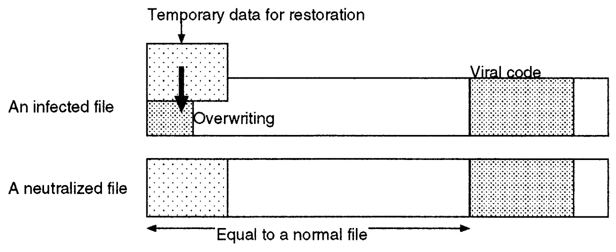


Fig. 3 Neutralization by an *antibody agent.*

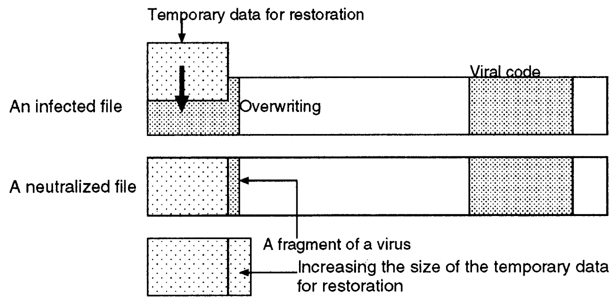

Temporary data for restoration
Viral code
An infected file
Overwriting
A neutralized file
A fragment of a virus
Increasing the size of the temporary data for restoration

Fig. 4 Increasing the size of the temporary data for restoration (called "adaptation").

a JUMP code or the entry point to the virus body. When the overwritten range exceeds the range changed by the virus, the neutralization not only prevents the virus to be executed, but also allows the original file to be executed normally (but a program which checks its own code may not be executed because the viral code remains at the end of the file). Figure 3 shows the neutralization. The neutralization allows a quick restoration even without the network.

When the size of alteration by the virus exceeds the size of the temporary data for restoration, however, the neutralized file may not be executed normally, because the fragment of a virus still remains. In this case, the size of the temporary data for restoration increases (corresponding to "adaptation" of the immune system), and hence it can neutralize the virus in the second encounter (Fig. 4). The initial size and the maximum size are set up beforehand by a system administrator (see Sect. 5.2).

If the *killer agent* detects an infected file, it makes the file unexecutable. At the same time, other computers on LAN is informed of the infection through the *control agent*.

The *control agent* has the following three functions: a compulsory activation of a *killer agent* when the *con-*

[†]One of the strategies used by the immune system is to recognize "non-self" by using "self" (MHC).

[††]The antibody prevents a virus from invading a cell by binding the virus.

[†††]The natural killer cell and killer T-cell destroy infected cells.

[††††]The helper T-cell is involved in controlling and regulating other types of cells of the immune system.

trol agent received a report of the infection from the infected computer on LAN; the surveillance of the execution for an *antibody agent*; and reporting a virus infection to other computers on LAN.

3.3 Recovery System

The *recovery system* is realized by a file replication similar to a recovery mechanism of multi-cellular organisms†. First, we make the following three assumptions; 1) the *immunity-based system* can detect all viruses which infect the "self"; 2) the communication between each agents is not forged; 3) all the computers are not infected all the same time.

The computer system must keep at least two same programs in order to recover from the damage. However, it is difficult for the actual system to maintain two (or more) same files because of the limited capacity. Even if it backs up all files, since the backed up files also have a risk of being infected, there is no point of backing up files in the same computer.

We propose to distribute executable system files (under "*windows*" directory) to two or more computers in LAN. Those files contained in the "self" information on one computer are backed up to other computers in LAN, and the computer can receive the backed up files whenever it requires. Since executable system files exist in each computer after installing OS, those files need not to be distributed (however, when the version of the OS or the OS itself differs from each other, it needs to be distributed). In this way, the *recovery system* fully uses the network and redundancy of executable system files.

The *copy agent* in the *recovery system* processes transmission and reception of a copy of the file in parallel with the *control agent*. The flow of the process is as below (see Fig. 5, too).

Step 1 When the *antibody agent* or the *killer agent* detects virus infection, they inform the *control agent* of the infection. The *control agent* (client) will inform other *control agents* (server) in LAN of the infection.

Step 2 The *control agent* (server), informed of the infection, investigates whether or not its own computer is infected (by a *killer agent*). At the same time, the *control agent* (server) investigates the file corresponding to the infected file (i.e. if the infected file is "`\\client\windows\MEM.EXE`," the corresponding file is "`\\server\windows\MEM.EXE`" where `client` is infected computer and `server` is an uninfected one) by activating the *antibody agent*. If the file is not infected, the *control agent* (server) informs a *control agent* (client) in the infected computer of the server name.

Step 3 The *control agent* (client) appends necessary information (the server name and the infected file name with the full path in the "self" information) to a list in the order that the *control agent* (client) receives the server name for a fixed period of time after virus detection.

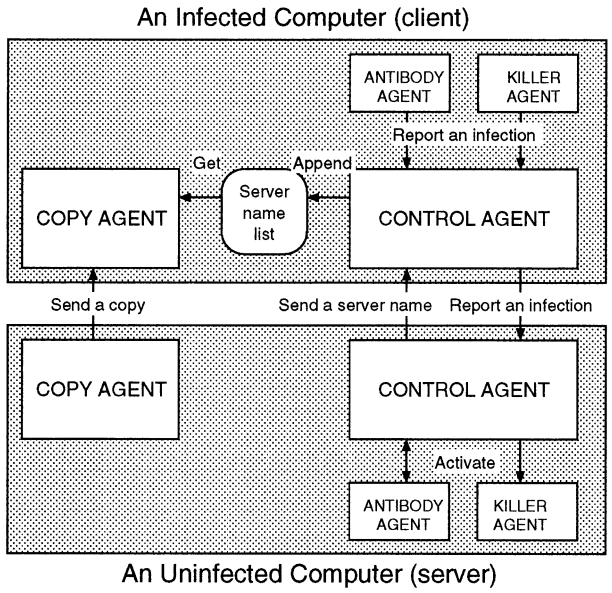


Fig. 5 The overview of the *recovery system*.

Step 4 The *copy agent* (client) gets the server name from the server name list, and receives the copy of the file from the *copy agent* (server). If the list is empty, the *copy agent* (client) waits for the arrival of a new data at the list.

Step 5 If the received file matches with the "self" information, the *copy agent* (client) copies the received file to the path indicated in the "self" information. When copying, the *copy agent* (client) changes some parameters (called "adaptation," see Sect. 3.2 for detail) in order to handle the second infection more quickly. If the received file does not match, then go to Step 4.

Step 6 After the fixed period of time from virus detection, the *control agent* (client) investigates the infected file by activating the *antibody agent*. If the file is still infected, the *control agent* (client) reports the failure of file restoration to a system administrator. The file recovery is terminated.

4. Experiments

In order to examine the validity of our system, we have implemented a prototype by JAVA^{TM} language (JAVA^{TM} Developers Kit 1.2.) which is a multi-platform language. We tested the prototype against some existing viruses, and compared the performance

†Multi-cellular organisms hardly suffers a fatal damage if the attack by virus is not hard, because the damage is recovered cell division.

Table 1 Comparison of the proposed system with commercial systems.

Virus Name	Size(byte)	Detection Stage				Recovery Stage				
		Commercial Sys.			Ours	Commercial Sys.			Neut.	Recov.
Scream3	about 7	S	S	S	S	F	P	F	S	S
Joker2	3	S	S	S	S	COM	COM	F	S	S
Commander Bomber	unknown	S	S	S	S	F	F	F	P	S
AP–605	666	S	S	S	S	COM	COM	COM	S	S
AIDS	13,952	S	S	S	S	F	F	F	F	S

with the present commercial systems. We experimented on LAN which connected four computers (whose OS is Windows 98^{TM}). As target files for virus detection, we choose the executable system files (programs under "*windows*" directory) of Windows 98^{TM}. (See [13] for similar experiments by Windows 95^{TM}.) Since the size of alteration by viruses was usually less than 10 Kbytes, we set the size of the temporary data for restoration to 10 Kbytes.

We used the following viruses[†]: `Scream3`, `Joker2`, and `AIDS`, `Commander Bomber`, and `AP-605`. These viruses are typical one from each infection types explained in Sect. 2.1.

Our system is compared with some of commercial systems focusing on detection stage and recovery stage. We selected three commercial softwares and tested them with their newest prescription in September, 1999. In our system, recovery stage is further divided into neutralization and file recovery (by copying from the other computers). Table 1 summarizes the experimental results on detection stage and recovery stage. S, P and F represent success, partial success, and failure respectively. COM means that only COM files are successfully recovered.

Since these five computer viruses have been already investigated, all of commercial systems successfully detected them. There is no commercial system (among those used in our experiments) that can successfully recover from all these five viruses. There are several reasons for the failure of recovering them. One is because a single prescription can remove only a single virus and the number of the prescriptions is less than that of viruses, although a single "signature" allows detection of dozens of distinct viruses [4]. Another reason is that viruses overwrite the head of the original host program with their viral code. Also, it can be pointed out that recovery succeeds more for COM files than for EXE files. This is because the infection mechanism for a COM file is simpler than that for an EXE file.

Virus detection by our system is successful for all five viruses, since these viruses alter files. Neutralization is not successful against `Commander Bomber` and `AIDS`. `Commander Bomber` takes its body into pieces of random size and overwrites the host program with them. Alteration by `AIDS` virus exceeds the data kept in the "self" information. However, the virus no longer infect, since the JUMP code to the main viral codes is already overwritten by the temporary data for restoration. File recovery (by copying from the other computers) succeeds against these five viruses.

In summary, our system can deal with viruses infecting programs for MS–DOS^{TM}. Other than viruses infecting those files, there are viruses that infect the system area (a boot sector) and PE(Portable Executable) file for Windows 95^{TM} or later. Our system may be extended against these viruses by treating the system area as a file and registering these code as the "self" information.

5. Discussion

Our prototype and the experiments raised some questions. They include; 1) why the "self" information should be executable system files under "*windows*" directory; 2) how long the size of the temporary data for restoration should be; 3) what kind of viruses our system can deal with in the future; and 4) the problems in the actual use. This section discusses these questions.

5.1 The "Self" in a Computer System

Our system used the checksum method in order to detect viruses. The checksum method can detect viruses infecting a file without the virus information ("signature"), since it is a technique for detecting the alteration of a file. However, it cannot distinguish whether the alteration is just a legitimate update or the invasion by a virus. The simple solution is to choose files whose alteration is not allowed as the target of detection. Although the files not allowed alteration differ from computer to computer, executable system files are rarely updated. Since the target of detection is only the executable system files, all alterations to those files are regarded as invasion.

If the "self" is limited to only executable system files, then the limitation causes many false negatives when programs other than executable system files are infected. In this case, the combination of the scanning method and the heuristics method may be used. For the security, the hybrid system integrating our method, the scanning method and the heuristics method becomes more secure than commercial anti-virus systems. Even

[†]Those viruses are collected from some Internet Web sites, and we use the names indicated in the sites. They may differ from ones used by commercial anti-virus softwares.

if the virus which can be detected neither by the scanning method nor by the heuristics method invades, they will be detected and recovered at the time of secondary infection to executable system files. Also, the detection of the second infection to the executable system files leads to the development of the prescription by informing vendors of the commercial anti-virus systems.

The scope of programs registered as the "self" will be determined by the trade-off between a false positive and a false negative. The trade-off is due to the requirements for the availability and the security of the computer system. Hence, the scope of the "self" is ultimately determined according to the security policy of each computer.

5.2 Adjustment of the Size of the Temporary Data for Restoration

Our system adapts to the environment by changing the size of the temporary data for restoration kept in the "self" information. If the infected files are not neutralized by *antibody agents* (they are removed and recovered as long as infection is detected), the size will be increased to prepare for the second encounter with the same virus. Determination of the size of the head part is a delicate problem which may be determined by the environments of the computer (whether or not the infection is highly possible). If the size is set to be maximum, the database of the "self" information becomes full backup and the system can recover only by neutralization, but doubles the used disk space.

5.3 Extension against Other Viruses

In this paper, we proposed the system to detect and remove the non-resident viral strains in memory which infect programs for MS–DOS^{TM}. Since our system can adjust the scope by choosing the "self," it may be possible to detect and remove new viral strains infecting a new file type, adding the files of the new file type to the "self" information. The following three types of viral strains may be treated by extending our system.

- First type is a viral strain which infects PE files. The structure of the PE file is similar to that of the programs for MS–DOS^{TM}. Then, the infection mechanism is classified into the same type as that of the program for MS–DOS^{TM}.
- Second type is a viral strain which infects a boot program in a boot sector. The infection mechanism is different from those infecting a file, because a space under possible infection is smaller than that of infected file by the file infection type. Hence, our system may fail to recover the infected boot sector.
- Third type is virus-like malicious programs (the "companion virus"[12] and "Trojan horse") which do not alter a host program. This type creates another file having the same name as the original one except for their extension[†]. Fortunately, we can find the malicious program by investigating the file list in the directory where the host program is located. Therefore, this type may be handled by using the file list as "self." However, it is difficult to know whether the addition is a legitimate installed application or an illegal invasion of the malicious program. The decision of registering the file list at the "self" information depends on the requirements for an availability and a security.

5.4 The Problems in Actual Use

A serious problem of our system may be protecting the database of the "self" information. If a computer virus attacks the database, our system can not detect and recover, or may lead to many false positives. The database should be encrypted and/or distributed over LAN to minimize such possibility.

Another problem is weakness against resident viruses in memory. Such viruses hook some of MS–DOS^{TM} interrupt handler (`Int 13h`, `Int 21h`, etc), and infect a program when accessed (i.e. when the interrupt handler is called for operating the file). Even if an *antibody agent* and a *copy agent* repair an infected file, the repaired file may be infected again. Moreover, if a "stealth virus" which conceal itself infects a program, our system cannot detect it. The method of detecting the stealth viruses only scan a memory for the "signature." Even if a scanning software detects it in memory, the computer must be rebooted to repair it under clean boot using clean floppy disk. File recovery in general is a common problem to all anti-virus systems.

6. Conclusion

The proposed method differs from most of commercial anti-virus systems, since it uses the information of the host program side rather than that of virus side ("signature"). The method differs from the so far proposed anti-virus system that uses the computer network for delivery of prescriptions, since our method uses the computer network to mutually back up the files existing in several computers.

By the experiments, we have shown that our system can successfully detect, neutralize and recover against some of viruses infecting programs for MS–DOS^{TM}. Moreover, the intensity of security may be raised further by incorporating the scanning method and the heuristic method into our system. However, our system requires a protection of the database of the "self" information and sophisticated control mechanism

[†]For example, if the original file (infected file) is "`MEM.EXE`," the virus creates "`MEM.COM`."

to prevent the secondary infection in the recovery process.

Acknowledgement

This work has been supported in part by the SCAT (Support Center for Advanced Telecommunications Technology Research) Foundation and a General Research Grant (C 10650431) by the Ministry of Education, Science and Culture.

References

[1] ISCA, "Isca 1999 computer virus prevalence survey," 1999. http://www.icsa.net/ services/ consortia/ anti-virus

[2] D. Chess, "The future of virus on the internet," Proc. 7th Virus Bulletin Conf., Oct. 1997. http://www.ar.ibm.com/ InsideTheLab/ Bookshelf/ scientificPapers/ Chess/ Future.html

[3] Symantec corp., Understanding heuristics: Symantec's bloodhound technology, 1997.

[4] J.O. Kephart, G.B. Sorkin, D.M. Chess, and S.R. White, "Fighting computer viruses," Scientific American, no.11, pp.88–96, Nov. 1997.

[5] J.O. Kephart, G. Tesauro, and G.B. Sorkin, "Neural networks for computer virus recognition," IEEE Expert, vol.11, no.4, pp.5–6, Aug. 1996.

[6] J.O. Kephart, "A biologically inspired immune system for computers," Proc. 14th Int. Joint Conf. on Artificial Intelligence, pp.20–25, Aug. 1995.

[7] S. Forrest, P. D'haeseleer, and P. Helman, "An immunological approach to change detection: Algorithms, analysis and implications," Proc. IEEE Symposium on Research in Security and Privacy, pp.110–119, May 1996.

[8] C. Pu, A. Black, C. Cowan, and J. Walpole, "A specialization toolkit to increase the diversity in operating systems," ICMAS Workshop on Immunity-Based Systems, 1996. http://www.sys.tutkie.tut.ac.jp/ ∼ishida/ IMBS96proc.html

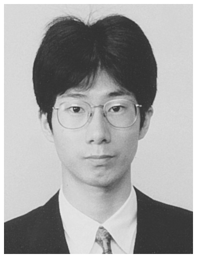
[9] Y. Ishida, "The immune system as a prototype of autonomous decentralized systems," Proc. Int. Symposium on Autonomous Decentralized Systems, pp.85–92, 1997.

[10] F. Cohen, "Computer viruses, theory and experiments," Proc. 7th DOD/NBS Computer & Security Conf., pp.22–35, 1987.

[11] E.H. Spafford, "Computer viruses—A form of artificial life ?," in Artificial Life II, Studies in the Sciences of Complexity, Addison-Wesley, 1991.

[12] V. Bontchev, "Possible virus attacks against integrity programs and how to prevent them," Proc. 2nd Int. Virus Bulletin Conf., pp.131–141, Sept. 1992.

[13] T. Okamoto and Y. Ishida, "A distributed approach to computer virus detection and neutralization by autonomoous and heterogeneous agents," Proc. 4th Int. Symposium on Autonomous Decentralized Systems, pp.328–331, March 1999.



Takeshi Okamoto received the degree of B.S. in Engineering from Ritsumeikan University, Kyoto, in 1996 and the degree of M.S. in Computer Science from Nara Institute of Science and Technology, Nara, in 1998. He is currently doctoral student at Department of Knowledge-based Information Engineering at Toyohashi University of Technology.

Yoshiteru Ishida received Ph.D. in Applied Mathematics and Physics from Kyoto University in 1986. He served as an assistant professor at Department of Applied Mathematics and Physics Kyoto University from 1983 to 1986, and at Division of Applied Systems Science from 1987 to 1993. From 1994 to 1998, he has been an associate professor at Graduate School of Information Science, Nara Institute of Science and Technology. Since 1998, he has been a professor at Department of Knowledge-based Information Engineering at Toyohashi University of Technology. During this period, he had been a visiting researcher at School of Computer Science, Carnegie-Mellon University (1986–1987), Department of Psychology, Carnegie-Mellon University (1993–1994) and Santa Fe Institute (1997–1998).