Implementation of Logging for Information Tracking on Network

Akihiko Maeta
Graduate School of Engineering, Tottori University
4-101 Koyama-Minami, Tottori 680-8550, Japan
s082050@ike.tottori-u.ac.jp

Kenichi Takahashi
Graduate School of Engineering, Tottori University
4-101 Koyama-Minami, Tottori 680-8550, Japan
takahashi@ike.tottori-u.ac.jp

Takao Kawamura
Graduate School of Engineering, Tottori University
4-101 Koyama-Minami, Tottori 680-8550, Japan
kawamura@ike.tottori-u.ac.jp

Kazunori Sugahara
Graduate School of Engineering, Tottori University
4-101 Koyama-Minami, Tottori 680-8550, Japan
sugahara@ike.tottori-u.ac.jp

***Abstract*—In recent years, information leakage cases happen frequently and are one of big problems. 70% of such a cases are caused by human factors such as mismanagements and/or careless operations. Such a factors can be removed by the control of information flow and/or the check of computer operations. Therefore, some studies to control information flows have been proposed. However, almost studies focus on single computer. Thus, it is insufficient in recent working situations which exchange a lot of information through network.**

In this paper, we propose a framework to track sensitive information flow on multiple computers at kernel. The framework hooks *system calls* which may cause the diffusion of information, maintains their logs, and uses their logs to trace information. As the result, we can trace information on multiple computers. It would be useful for the control of the operations which uses sensitive information.

I. Introduction

In recent years, the opportunities to manage the information as not only papers but also digital data on computers are increasing. This situation enables us to easily diffuse information. For example, we can easily share information by using a mailing list, a shared folder, a usb memory and so on. This, however, increases the risk of information leakage. For example, we may share a wrong file including confidentiality information instead of an ordinary file by mis-operation. It will cause information leakage.

In the report from NPO Japan Network Security Association on 2011 [1], 70% of information leakage cases are caused by mismanagement and/or careless operation, such as, the mistake of To: header in an e-mail, the an attached file. Information leakage caused by technical factors on a computer, such as malware, illegal access, security holes, are about only 5%. Thus, it is important to cut information leakage cases caused by the carelessness of human being. Therefore, some studies to trace information flows have been proposed.

DF-Salvia[2] and Privacy-aware OS Salvia[3] propose an operating system which enables us to track information flow. [4] and [5] gives a warning when a user tries to copy a file to a usb memory. [6] and [7] tries to prevent information leakages from e-mails by the check of a superior. Their studies, however, focus on single computer. Thus, they are insufficient in recent working situations which exchange a lot of information through network.

In this paper, we proposes a framework to track sensitive information on multiple computers at kernel. The framework hooks *system calls* which may cause the diffusion of information, maintains their logs, and uses their logs to trace information. As the result, we can know information flows on multiple computers. It would be useful for the control of the operations which may cause information leakage.

The remainder of this paper is structured as follows. Section II describes when the diffusion of sensitive information occurs at kernel and presents the problems when tracking sensitive information. Section III shows implementation details to solve the problems shown in Section II. Section IV examines our framework and concludes the paper at Section V.

II. Tracking Sensitive Information

We propose a framework to track sensitive information on multiple computers. The sensitive information is highly confidential information such as credit card number, personal information and so on. Since such a information is usually managed as a file unit, we focus on files as the objects for tracking. Further, we should trace the sensitive information without depending on applications. All the access to files is gone through kernel level operations. Therefore, we can trace the diffusion of all sensitive information by recording file operations at kernel level. Figure 2 illustrates the overview of the diffusion of sensitive information at kernel.

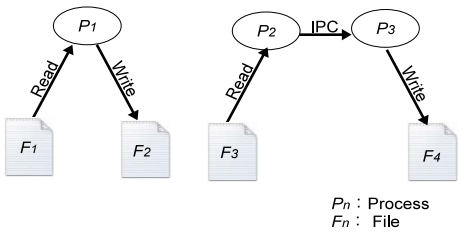

Fig. 2. Diffusion of Sensitive Information

978-1-4799-2845-3/13/$31.00 ©2013 IEEE

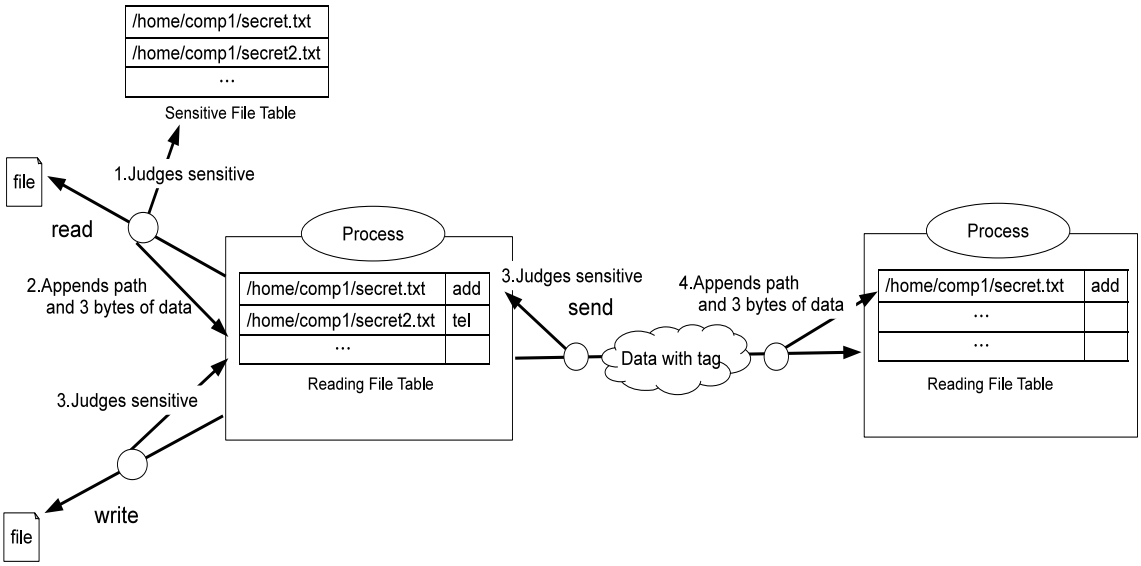

Fig. 1. Overview of the Implementation

As shown in Figure 2, all the diffusion of sensitive information is caused through *process* by read, write and IPC (InterProcess Communication). For example, when process P_1 writes out information to a file F_2 after P_1 reads a file F_1 which has sensitive information, the sensitive information of F_1 may be written in F_2. Here, the process can know only reading/writing a file. It is difficult for a process at kernel level to know which information is read from and/or written in a file. Therefore, we trace the possibility of the diffusion of sensitive information.

The right illustration of Figure 2 is the overview of the diffusion of sensitive information caused by IPC. In this example, after the process P_2 reads a file F_3, P_2 communicates with the process P_3 by IPC. Then, the information in F_3 may propagate to P_2. Therefore, the information which P_3 writes down in a file F_4 may include the information in F_3. Thus we have to monitor operations related to read, write and IPC for tracking sensitive information. There are the following problems to be solved for tracking sensitive information at kernel level:

Problem 1: Distinction of files which includes sensitive information,

Problem 2: Judgement of processes which may read sensitive information,

Problem 3: Judgement that information received by IPC may be sensitive information or not.

The first problem is the distinction of files which includes sensitive information. If a file has not any sensitive information, it is meaningless to trach the file. Therefore, we have to distinguish the information read by a process may include a sensitive information. In order to distinguish it, we prepare a *sensitive file table*, which manages files including sensitive information. By refering the sensitive file table, the system judges whether a file includes sensitive information or not.

The second problem is the judgement of processes which may read sensitive information. When a process writes some information into a file, we have to know the information may be sensitive. That is, the system has to know the process has read a file including sensitive information in a past. We introduces a *reading file table* to manage files a process has read. When a process reads a file, the file information is added in the reading file table. When the process writes information into other file, the system can judge the information may include sensitive information by seeing the reading file table.

The last problem is the judgement whether information received by IPC may be sensitive information or not. The diffusion of information on network are caused by IPC. If a process receives information by IPC from other process reading sensitive information, the information may include the sensitive information. Then, the reading file table of the receiver process has to be updated. The receiver process, however, does not know the information received is sensitive or not. Therefore, the sender process informs it as a *tag* to the receiver process. The receiver process confirms that the information received is sensitive or not by the tag.

III. Implementation

We implemented a system for tracing sensitive information on linux kernel 2.6.22[8]. The targets of tracing sensitive information are file operation and IPC, especially socket communication. The file operation and IPC are implemented as *system calls* in kernel unit. Therefore, the system hooks *system calls* and checks the diffusion of sensitive information. We focus on read/write and sendto/recvfrom system call for tracing sensitive information. Figure 1 shows the overview of the system.The system consists of *sensitive file table*, *reading file table* and *tag* with transmission data.

A. Sensitive File Table

Information has been diffused when a process writes data into a file after the process reads information from a file. For tracing sensitive information, the system has to judge the

```
16:03:37 PC1 kernel: File Registered : UID:500 FILE:/home/comp1/d.txt
16:03:42 PC1 kernel: Read File : PID:3752 FILE:/home/comp1/d.txt
16:03:43 PC1 kernel: Write File : PID:3752 FILE:/home/comp1/e.txt(/home/comp1/d.txt) DEV:800002
16:03:43 PC1 kernel: File Registered : UID:500 FILE:/home/comp1/e.txt
16:04:36 PC1 kernel: Read File : PID:3854 FILE:/home/comp1/e.txt
16:04:36 PC1 kernel: Send File : PID:3854 send 127.0.0.1:37685 to 127.0.0.2:10000 FILE:/home/comp1/e.txt
```

(a) Logs in Computer PC_1

```
16:04:36 PC2 kernel: Receive File : PID:3850 receive 127.0.0.2:10000 from 127.0.0.1:37685 FILE:/home/comp1/e.txt
16:04:36 PC2 kernel: Write File : PID:3850 FILE:/home/comp2/recv.txt(/home/comp1/e.txt) DEV:800002
16:04:36 PC2 kernel: File Registered : UID:500 FILE:/home/comp2/recv.txt
16:05:21 PC2 kernel: Read File : PID:3871 FILE:/home/comp2/recv.txt
16:05:21 PC2 kernel: Write File : PID:3871 FILE:/home/comp2/m.txt(/home/comp2/recv.txt) DEV:800002
16:05:21 PC2 kernel: File Registered : UID:500 FILE:/home/comp2/m.txt
```

(b) Logs in Computer PC_2

Fig. 3. Examination Results

information read by a process includes sensitive information or not. Its judge is done by using *sensitive file table*. In our system,the sensitive file table has to be managed in the kernel unit. Because we the system traces the diffusion of sensitive information at kernel level. Therefore, we implements functions markSecret by using *proc* filesystem that allows us to access and change the data in kernel from the user mode.

When a user wants to treat a file "/home/comp1/secret.txt" a sensitive information, the user calls

`markSecret(ADD,/home/comp1/secret.txt)`

from a console. Then, the path "/home/comp1/secret.txt" are appended in the sensitive file table. Thus, we can manage the sensitive file table from the user mode. When a process reads a file, the system starts tracing the file if and only if the path of the file is included in the sensitive file table.

B. Reading File Table

When a process writes sensitive information into a file, the sensitive information is diffused. It is, however, difficult to judge what information is written in the file at kernel level. Therefore, we prepare a *reading file table* in each processes. A process records list of a file that the process read in a past, and 3 bytes from the head of data the process reads. In linux kernel, various information of each process has been managed in *task_struct* structure. Therefore, we prepare a reading file table in *task_struct*.

Reading/writing a file is realized by `read/write` system call in linux. Therefore, we appended program codes to operate reading file table into read/write system call. In our implementation, the read system call, first, gets the file path from *File Descripter*, and checks the file path is listed or not in the sensitive file table. Then, the read system call puts the file path and 3 bytes from the head of data the process read into the reading file table.

Write system call, we first checks any information is listed in the reading file table or not. If not listed anything, it does not do any special behavior since the process does not deal with sensitive information. If listed, the write system call compares 3 bytes from the head of data the process writes with all 3 bytes listed in reading file table. If they are matched, the data written in a file may includes sensitive information. Therefore, the write system call puts the file path into sensitive file table. Thus, the system can trace the diffusion of sensitive information in file operations.

C. Tag for Data Transmission

As explained in previous section, it is also difficult to know what data is transmitted between processes in IPC at kernel level. Therefore, we attached a *tag* with transmitted data from a sender to a receiver process. Figure 4 is the structure of a tag with transmitted data. The tag with transmitted data is structured in order of `PATH_SIZE`, the length of the path, path and data transmitted.

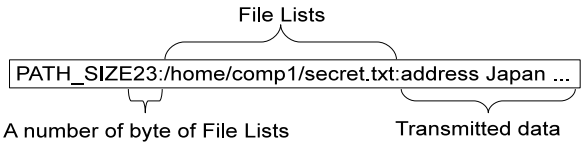

Fig. 4. Structure of a Tag with Transmitted Data

Here, we focus on only socket communications as IPC. The socket communications is realized by `sendto/recvfrom` system call. The sendto system call, first, checks the reading file table, and connects tag with the transmitted data if the data may include sensitive information. Then, the sender process transmits connected one to a receiver process. The receive process receives it by the recvfrom system call. The recvfrom system call checks that a tag is attached with the received data or not. If a tag is attached, the receiver process has to deal it as sensitive information, thus, puts the list in the tag into own reading file table. Thus, the system can trace the diffusion of sensitive information among multiple computers.

IV. Examination

We examine the diffusion of sensitive information by file operations and socket communications can be traced. In this examination, we prepared two computers PC_1 and PC_2 which

978-1-4799-2845-3/13/$31.00 ©2013 IEEE

IP address are 127.0.0.1 and 127.0.0.2; used *copy* command for file operations; and prepared a socket communication program. In the socket communication program, one side reads a file and transmits it to other side; the other side receives it and writes it down on a file. The examination results were conducted on the following steps (Figure 5):

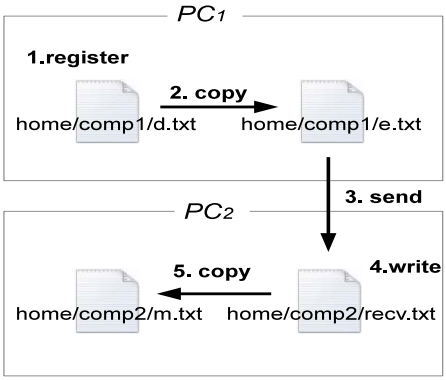

Fig. 5. Examination Steps

1) "/home/comp1/d.txt" is registered as a file including sensitive information in computer PC_1,
2) "/home/comp1/d.txt" is copied to "/home/comp1/-e.txt" in PC_1,
3) PC_1 reads "/home/comp1/e.txt" and transmits it to computer PC_2,
4) The process in PC_2 which received data from PC_1 writes it down on "/home/comp2/recv.txt,"
5) "/home/comp2/recv.txt" is copied to "/home/comp2/-m.txt" in PC_2.

The results in this examination are shown in Figure 3. The log consists of time, computer name, operation, user ID (UID) or process ID (PID), and operational detail, in order from the left to the right.

The first line of logs in PC_1 shows "/home/comp1/d.txt" is registered in the *sensitive file table* as a file including sensitive information. Next lines 2 to 4 were generated from step 2. We can confirm the process 3752 reads "/home/comp1/d.txt" from line 2, and writes it to "/home/comp1/e.txt" from line 3. Here, since the process 3752 reads a file "/home/comp1/d.txt" registered in the sensitive file table, "/home/comp1/e.txt" should be also registered in the sensitive file table. We can confirm it in line 4. As the result, the diffusion of sensitive information caused by file operations has been traced.

After that, PC_1 reads "/home/comp1/e.txt" and transmits it to a computer PC_2 at step 3. Therefore, their logs are written in line 5 and 6 in PC_1. Here the tag "/home/comp1/e.txt" is attached with transmitted information, because "/home/comp1/e.txt" is listed in the sensitive file table. Then, the process 3850 in computer PC_2 receives data, and writes it down on "/home/comp2/recv.txt" at step 4. Here, since the process 3850 notices the tag attached, "/home/comp2/recv.txt" is also registered in the sensitive file table in PC_2. We can confirm them from the line 1 to 3 of logs in PC_2. Thus, the diffusion of sensitive information among multiple computers can be traced.

Remained logs, the line 4 to 6 in PC_2, were generated from step 5. As the same with step 2, "/home/comp2/m.txt" is registered in the sensitive file table. As the result, we can confirm the sensitive information in "/home/comp1/d.txt" may be diffusing to "/home/comp2/m.txt" through "/home/comp1/e.txt" and "/home/comp2/recv.txt." Threfore, we are able to care the dealing with these files, for example, when they are transmitted to outside of a company, copied to a usb memory and so on.

V. Conclusion

In this paper, we implemented the tracking system of sensitive information among multiple computers in linux kernel. In this system, we introduced *sensitive file table*, *reading file table* and tags attached with tranmitted information for the management of sensitive information. Examination results shows our system can trace the diffusion of sensitive information caused by file operation and socket communications.

The future works include to implement the tracing mechanism into other IPC, such as pipe, shared memory, memory mapping and so on.

Acknowledgment

This research has been supported by a Grant-in-Aid for Young Scientists (B), 23700098 and Grant-in-Aid for Scientific Research B, 23300027.

References

[1] NPO Japan Network Security Association,Information Security Incident Survey Report 2012, http://www.jnsa.org/result/incident/2012.html

[2] Shozo Ida, Yusuke Kawashima, Takehiro Kashiyama, Eiji Takimoto, and Koichi Mouri,Design and implementation of DF-Salvia which provides access control based on data flow, 2011 Information Processing Society of Japan. The 73rd national convention lecture memoirs, Vol2011No1pp511-513(3 2011)

[3] Kaji Teruyuki, Iwanaga Masayuki, Mouri Koichi, A construction of Privacy-aware OS Salvia based on Linux Security Module, Information Processing Society of Japan. The 71rd national convention lecture memoirs, Vol3No1pp365 - 366(3 2009).

[4] Iwanaga Masayuki, Mouri Koichi, File access control based on target devices for preventing data leakage, Information Processing Society of Japan. The 71rd national convention lecture memoirs, Vol3No1pp355 - 356(3 2009).

[5] Toshihiro Tabata, Satoshi Hakomori, Kei Ohashi, Shinichiro Uemura, Kazutoshi Yokoyama and Hideo Taniguchi, Tracing Classified Information Diffusion for Protecting Information Leakage 2009 Information Processing Society of Japan, Vol50No9pp2088-2102(9 2009)

[6] Active! gate. http://www.transware.co.jp/product/ag/sv.html

[7] GRIDY Mail Powered by NIFTY Cloud. http://knowledgesuite.jp/service/niftymail/niftymail-detail01.html

[8] Daniel P. Bovet, Marco Cesati (11,2005), Understanding the Linux Kernel, 3rd Edition , O'Reilly Media.