Improvement and Evaluation of a Function for Tracing the Diffusion of Classified Information on KVM

Hideaki Moriyama, Toshihiro Yamauchi, Masaya Sato, and Hideo Taniguchi

Abstract The increasing amount of classified information currently being managed by personal computers has resulted in the leakage of such information to external computers, which is a major problem. To prevent such leakage, we previously proposed a function for tracing the diffusion of classified information in a guest operating system (OS) using a virtual machine monitor (VMM). The tracing function hooks a system call in the guest OS from the VMM, and acquiring the information. By analyzing the information on the VMM side, the tracing function makes it possible to notify the user of the diffusion of classified information. However, this function has a problem in that the administrator of the computer platform cannot grasp the transition of the diffusion of classified processes or file information. In this paper, we present the solution to this problem and report its evaluation.

1 Introduction

Personal computers are currently managing increasingly large amounts of classified information, and leakage of this information to external computers has become a major problem. Such leakage often occurs inadvertently and through mismanagement. In addition, cyber-attacks aiming to steal classified information have become increasingly sophisticated. To prevent information leakage, users need to understand the risks associated with classified information. Furthermore, as complete prevention of cyber-attacks is difficult, it is important to mitigate the damage incurred by users by detecting the transfer of classified information from their computers.

Hideaki Moriyama
National Institute of Technology, Ariake College, 150 Higashihagio-Machi, Omuta Fukuoka, Japan
e-mail: hideaki@ariake-nct.ac.jp

Toshihiro Yamauchi, Masaya Sato, and Hideo Taniguchi
Graduate School of Natural Science and Technology, Okayama University, Okayama 700-8530, Japan

To determine the status of classified information stored on a computer and to manage the resources that contain such information, we previously proposed a function for tracing the diffusion of classified information in a guest OS using a virtual machine monitor (VMM), and implemented on kernel-based virtual machine (KVM) [1] [2] (i.e., a tracing function). The tracing function manages any file or process with the potential to diffuse classified information in the guest OS. Classified information can be diffused by any process that involves opening a sensitive file, reading its content, communicating with another process, or writing the file's content to another file. The proposed tracing function operates as follows. First, the administrator registers any file containing classified information as a file that will potentially leak classified information (i.e., a managed file). If a process executes specific operations on such a file, it may cause the classified information to leak. Therefore, the tracing function registers this process that has the potential to diffuse classified information (i.e., a managed process). By registering the process that operates the managed files and files or processes that are generated by the managed process, the administrator can detect classified information leaks. The tracing function is implemented by modifying a VMM. Therefore, this function can be implemented without modifying the source code of the OS. Further, attacks targeting this function will be difficult to execute because a VMM is more robust than an OS. However, the tracing function may have large processing overheads because it hooks all system calls on the VM and registers the processes and files containing classified information. We analyzed the processing performance of the tracing function [1] [2] in detail and identified that the processing of the system call exit involves large overheads, as demonstrated in our previous study [3]. Moreover, we presented a policy for efficient management to reduce these overheads and reported on its evaluation.

Although the tracing function reported by Fujii et al. [1] [2] can detect the process and files that has potential to diffuse classified information, the administrator cannot grasp such information at any time. Specifically, the administrator cannot grasp the list of the managed processes and files from the start of the tracing function to the present, as well as the list of the managed processes and files currently registered.

In this study, we developed a solution to the above mentioned problems. In addition to the method for reducing the processing overheads incurred in outputting information to the system log, which we demonstrated in our previous study [3], we implemented a function that outputs information regarding processes and files to the system log when they are not registered. Moreover, we developed a function that outputs a list of the managed processes and files from the start of the tracing function to the present, as well as a list of the managed processes and files currently registered.

In addition to the method for reducing the processing overheads presented in our previous study [3], we report on its evaluation using the system call of the file operation and benchmarks.

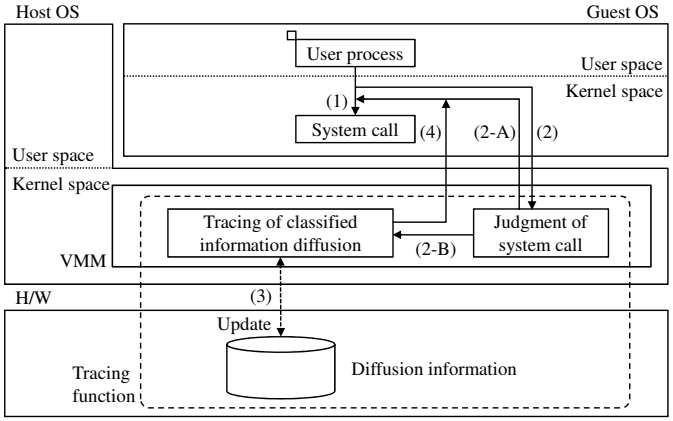

Fig. 1 Overview of the tracing function

2 Function for Tracing Diffusion of Classified Information

2.1 Overview

Fig. 1 shows an overview of the tracing function. When a user program in the guest OS requests a system call, the tracing function hooks the system call entry (via the SYSCALL instruction) and the system call exit (via the SYSRET instruction) using the hardware breakpoint (Step 1 in Fig. 1). Therefore, the tracing function can hook the system call using the VMM by detecting debug exceptions in the guest OS (Step 2 in Fig. 1). If the hooked system call is related to the diffusion of classified information (Step 2-B in Fig. 1), the tracing function collects the information needed to trace the diffusion (Step 3 in Fig. 1). Thereafter, control is returned to the guest OS and the system call process continues (Step 4 in Fig. 1). If the hooked system call is unrelated to the diffusion of classified information, control is returned to the guest OS (Step 2-A in Fig.1), and the system call process continues.

2.2 Process Flow

Fig. 2 shows the process flow of the tracing function. After the process moves to the VMM side, the tracing function process branches depending on whether the exception that occurred is to be handled via SYSCALL instruction or SYSRET instruction.

By hooking the SYSCALL instruction, the tracing function obtains the system call number, page table information, and value of the file descriptor. On the other hand, by hooking the SYSRET instruction, the tracing function obtains the system

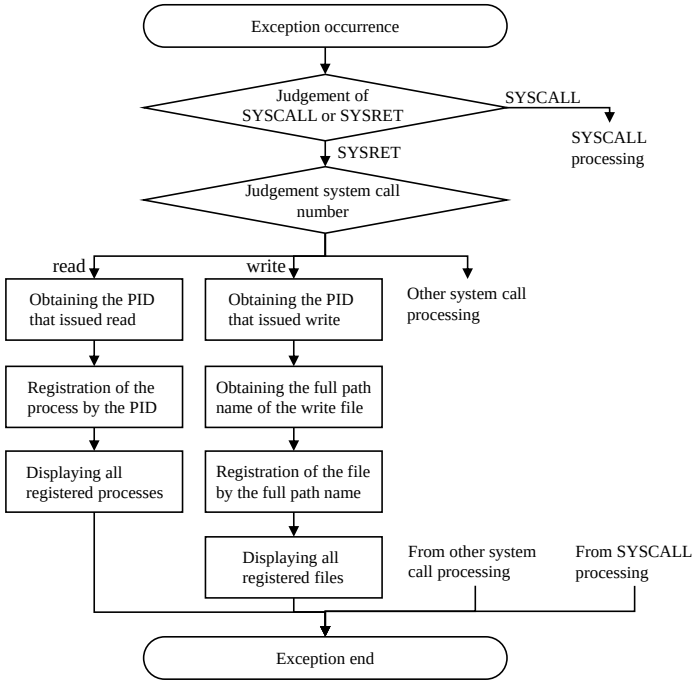

Fig. 2 Process flow of the tracing function

call number, return value of the success or failure of the system call, and details of the file handled by the system call. At this time, the tracing function determines whether the hooked system call is related to the diffusion of classified information; in which case, it collects the information needed to trace the diffusion. Control is then returned to the guest OS and the system call process continues. If the tracing function registers the managed file or process, it records the pathname of the destination file, inode number, the command name that caused the diffusion, and process ID. Furthermore, the tracing function outputs this information to the system log (/var/log/messages).

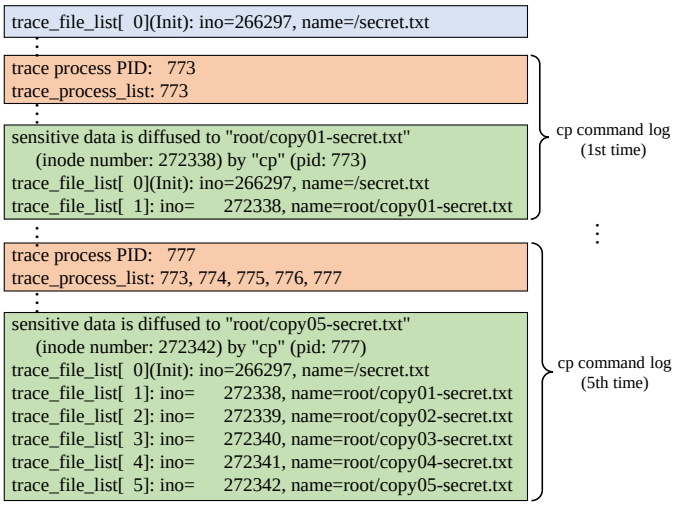

Fig. 3 Example of the log of the diffusion of classified information

3 Grasping the Potential Diffusion of Classified Information

3.1 Problem with the Current Method

When the administrator of the guest OS or the administrator of the computer confirms the diffusion of classified information, satisfaction of the following two requirements is necessary.

Requirement 1: The administrator must grasp the list of the managed processes and files from the start of the tracing function to the present.

Requirement 2: The administrator must grasp the list of the managed processes and files currently registered.

In the execution of the service, by satisfying Requirement 1, the administrator can grasp the classified information that is referred to, updated, or newly generated by the service. This would enable the administrator to determine whether the service is handling the classified information as intended. Moreover, by satisfying Requirement 2, the administrator can grasp the current classified information.

The previous tracing function outputted the list of all the managed processes that would potentially leak information from the start of the tracing to the present whenever a new process was registered. Similar to the registration of a new process, the previous tracing function outputted the list of all the managed files that would potentially leak from the start of the tracing to the present, whenever a new file was registered. Fig. 3 shows an example of this log. Fig. 3 is an example of a log in

which /secret.txt (inode=266297) is registered as managed file that is potentially being leaked; the file has been duplicated five times using the cp command. In the fifth execution of the cp command, the tracing function registers a newly managed process (PID = 777) when the read system call occurs, and the tracing function outputs the information regarding the five managed processes (PID = 773, 774, 775, 776, 777) as a system log from tracing start to end. Moreover, the tracing function registers the newly managed file copy05-secret.txt (inode = 272342) when the write system call occurs, and the tracing function outputs the information regarding the six managed files (inode = 266297, 272338, 272339, 272340, 272341, 272342) as a system log from tracing start to end.

Therefore, the previous tracing function required the editing process based on the log of all diffusions of classified information to satisfy Requirement 1. Furthermore, the previous tracing function cannot satisfy Requirement 2 because it does not output the log when a process or a file is no longer being managed.

3.2 Improved Method to Grasp the Potential Diffusion of Classified Processes/Files

To satisfy Requirements 1 and 2, in this study, we implemented the following two improvements:

Improvement 1: The tracing function outputs the log messages when a process becomes exits.
Improvement 2: The tracing function outputs the log messages when a file is removed.

Additionally, we implemented the following two improvements, as shown in our previous study [3],

Existing Improvement 1: When the tracing function detects a newly registered process, it only outputs this process's information log.
Existing Improvement 2: When the tracing function detects a newly registered file, it only outputs this file's information log.

These improvements can reduce the processing overheads in the tracing function. Therefore, in addition to the above two improvements and two existing improvements, we also implemented the following two improvements:

Improvement 3: A processing function that integrates all classified information logs.
Improvement 4: A processing function that edits all classified information logs.

Consequently, Improvement 3 can satisfy Requirement 1, and Improvement 4 can satisfy Requirement 2.

Fig. 4 shows an example of the classified information log following implementation of the two new improvements and the two existing improvements. As in Fig.

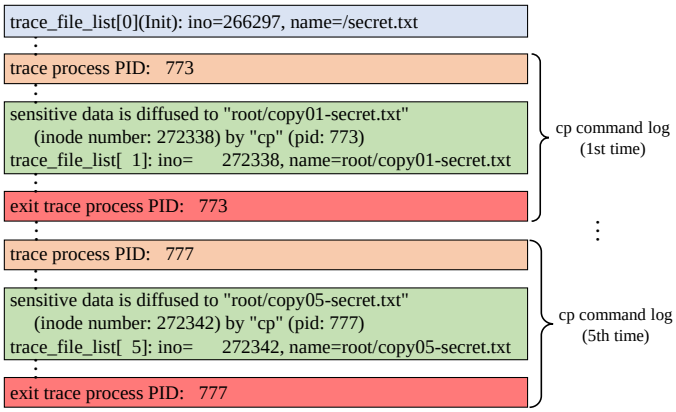

Fig. 4 Example of the classified information log of the improved tracing function

3, Fig. 4 is an example of a log in which /secret.txt (inode = 266297) is registered as a managed file, and the file is duplicated five times using the cp command. In the fifth execution of the cp command, the tracing function registers a newly managed process (PID = 777) and outputs only this process information as a system log. Moreover, the tracing function registers the newly managed file copy05-secret.txt (inode = 272342) and outputs only this file information as a system log. When the fifth execution of the cp command is completed, the tracing function also outputs the finished process (PID = 777) information as a system log using Improvements 1 and 2.

Note that the processing function of Improvement 3 that integrates all classified information logs can be achieved by integrating all logs. Therefore, editing processing in the previous tracing function is unnecessary. The processing function for editing all logs shown in Improvement 4 can be realized by excluding the finished process or the removed file information shown in Improvements 1 and 2 from the classified log information obtained by the existing Improvements 1 and 2.

4 Performance Evaluation

4.1 Perspective of the Evaluation

We reduced the processing overheads by implementing existing Improvements 1 and 2 and provided a basic performance evaluation in our previous study [3]. In this study, we implemented Improvements 1 and 2 in addition to the existing improve-

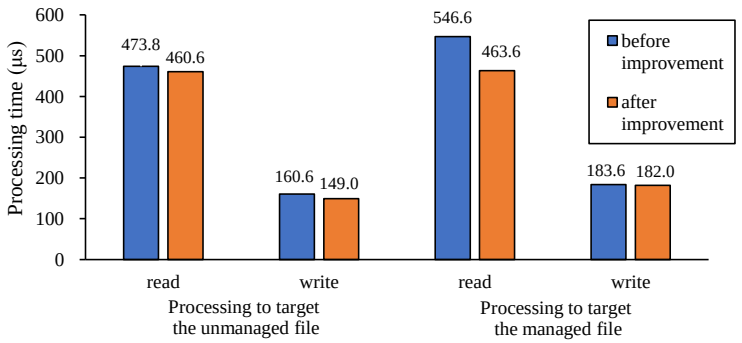

Fig. 5 Processing time of the read/write system calls

ments and performed a detailed evaluation using a file operation system call and a benchmark for file access.

4.2 File Operation System Call

To evaluate file operation, we measured the processing time of the read and write system calls. This operation obtains data from a 100 kB file by using the read system call and writes these data to another file using the write system call. The range of the measurement time is the processing time of the tracing function from the start to the end in each system call. We measured each case in which the tracing function registers the managed file and the unmanaged file. Moreover, we measured each case, using both the unimproved and improved tracing functions.

The measurement results are shown in Fig. 5. The measurement results demonstrated that the improvement effect was significant in the read system call. The processing time of the read system call for the target managed file was 546.6 μs before the improvement and 463.6 μs after the improvement. In summary, using the improvement, the processing time was reduced by 83.0 μs.

However, the improvement effect was not significant in the write system call. The processing time of the write system call for the target managed file was 183.6 μs before the improvement and 182.0 μs after the improvement. In summary, using the improvement, the processing time was reduced by 1.6 μs.

From the analysis of the processing overheads of the tracing function in our previous study [3], we established that, using the existing improvements, the effectiveness of reducing the processing overheads in the read system call was more significant than that of the write system call. The results of this measurement is explained by the analysis conducted in our previous study [3].

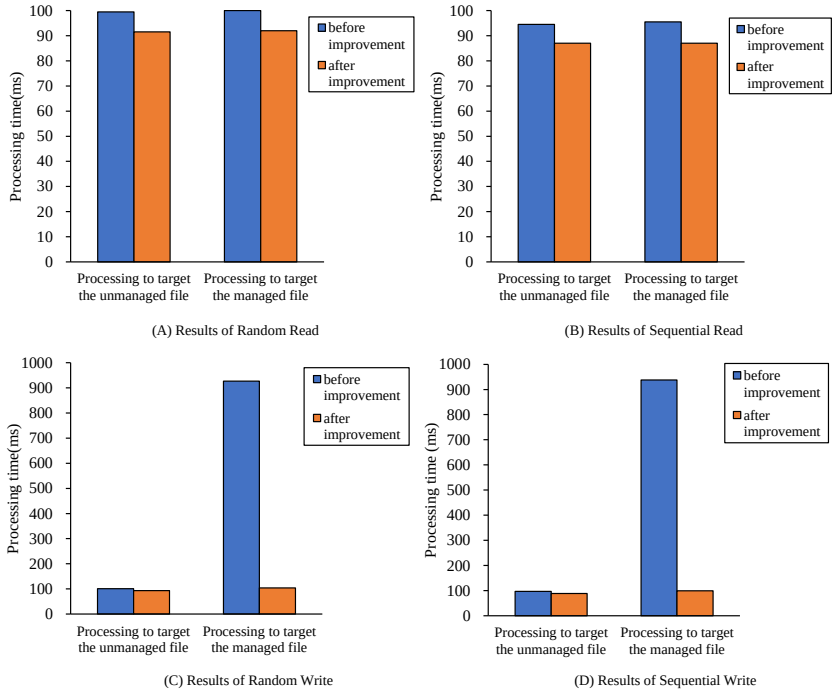

Fig. 6 Processing time using the fio benchmark

4.3 Benchmark for File Access

To evaluate file access performance, we used the fio (Flexible I/O Tester) benchmark. We measured the processing time using four types of file access patterns: Random Read, Random Write, Sequential Read, and Sequential Write. We prepared 1,000 4 kB files and accessed them using a block size of 4 kB. We measured each case in which the tracing function registers the classified and the unmanaged file. Moreover, we measured each case using both the unimproved and improved tracing functions.

The measurement results obtained using the fio benchmark are shown in Fig. 6. The measurement results of Random Read are shown in Fig. 6-(A). The effectiveness of reducing the processing overheads was not significant in each case. For example, the processing time of the target managed file was approximately 100 ms before the improvement and approximately 92 ms after the improvement. Therefore, using the improvement, the processing time was reduced by approximately 8 ms.

Additionally, the same characteristic as in Fig. 6-(A) is observed in Fig. 6-(B). The processing time of the target managed file was approximately 95 ms before the

improvement and 87 ms after the improvement. Hence, using the improvement, the processing time was reduced by approximately 8 ms, which is the same as (A).

Therefore, in the case of the Read access pattern, the reduction in processing time was less than 10%, and the effectiveness of the improvement is not significant, neither for Random nor Sequential, and whether the tracing file targets the managed nor the unmanaged file.

The measurement results of Random Write are shown in Fig. 6-(C). The effectiveness of reducing the processing overheads was significant when the tracing function targeted a managed file. For example, the processing time of the target managed file was approximately 927 ms before the improvement and approximately 104 ms after the improvement. Thus, using the improvement, the processing time was reduced by approximately 823 ms. However, the effectiveness of reducing the processing time was not significant when the tracing function targeted an unmanaged file. Next, the processing time was approximately 95 ms before the improvement and approximately 87 ms after the improvement. Therefore, using the improvement, the processing time was reduced by approximately 8 ms.

Additionally, the same characteristic as in Fig. 6-(C) is observed in Sequential Write, as shown in Fig. 6-(D). For example, the processing time of the target managed file was approximately 938 ms before the improvement and approximately 99 ms after the improvement. That is, using the improvement, the processing time was reduced by approximately 839 ms. The processing time was approximately 97 ms before the improvement and approximately 88 ms after the improvement. Hence, using the improvement, the processing time was reduced by approximately 9 ms.

Therefore, in the case of the Write access pattern, the reduction in processing time was approximately 89%, and the effectiveness of the improvement in targeting the managed file is significant, both for Random and Sequential. Meanwhile, the reduction in processing time was approximately 8%–9%, and the effectiveness of the improvement in targeting the unmanaged file is not significant.

Considering the measurement results, the effectiveness of the improvement in the Read access pattern for the managed file was insignificant and that in the Write access pattern was substantial. The effectiveness of the improvement in the Read access pattern appears insignificant in the comparison of the processing time before and after the improvement because the registration of the managed process occurred once. However, the effectiveness of improvement in the Write access pattern appears substantial in the comparison of the processing time before and after the improvement because the registration of the managed files occurred 1,000 times.

5 Related Work

Hizver et al. proposed a method for improving the performance of VM monitoring [4, 5, 6]. To reduce the performance degradation on a virtual machine introspection (VMI), they proposed a method for monitoring at regular intervals, instead of monitoring constantly. However, it has been pointed out that missed detection

may occur when monitoring at regular intervals. Similarly, Shi et al. achieved performance improvement by setting the EPT protection for monitoring at regular intervals [5]. In our proposed method, missed detection does not occur because the monitoring of the diffusion of classified information is constant. Additionally, we achieved performance improvement by reducing the output information to a log file. We did not compare about the evaluation of the performance, because these methods differ from our proposed method in the purpose of the system and the target environment.

Zhan et al. proposed a method for the fine-grained control flow integrity verification of a virtual machine (VM) to satisfy the performance requirements in actual operation [6]. Their method recommends, detecting the code execution in page units and comparing it with the correct processing flow, rather than detecting a branch, to prevent the VMM from being called frequently and to suppress performance degradation. However, the detection accuracy of this method is lower than that of branch detection. In our proposed method, we took care not to reduce important classified information during the improvement.

Jia et al. proposed a method to guarantee the integrity of VMM program code and the validity of data, considering that the VMM used by the VMI and the host OS may be damaged by an attack in a cloud environment [7]. Their proposed method is based on the premise of VMM safety.

Enck et al. proposed a system for information-flow tracking on Android [8]. Their system tracks information flow in Android by taint analysis using modified libraries. In contrast to this study, the proposed method does not require modification of programs running on the VM.

Ji et al. proposed a system to investigate attacks using information flow tracking [9]. Their system achieved low overheads by recording system call events and accurate monitoring using on-demand process replay. Although the proposed method collects all information required for tracing the diffusion of classified information, suppressing the log output reduces unnecessary performance degradation. Moreover, owing to the on-demand log display function, the system manager can analyze the diffusion using the log information.

6 Conclusion

In this paper, we described the improvement and evaluation of a function for tracing the diffusion of classified information on KVM. First, we clarified the requirements of the tracing function when the administrator of the guest OS or the administrator of the computer confirms the diffusion of classified information. Moreover, we clarified the problems regarding the processing and outputting of a log. To address these problems and satisfy the requirements, we proposed four improvement methods. In the evaluation section, we measured the processing time of the tracing function to evaluate the effectiveness of the improvements. In the measurement results obtained using a file operation system call, the improvement effect was significant in the read

system call and the processing time was reduced by 83.0 μs. Moreover, in the measurement results obtained using the fio benchmark, the reduction in processing time was approximately 89% in the case of the Write access pattern; further, the effectiveness of the improvement in targeting the managed file is significant.

In our future studies, we will reduce the overheads related to the construction of a full pathname.

Acknowledgements This work was partially supported by JSPS KAKENHI Grant Numbers 19H04109 and 19K20246.

References

1. S. Fujii, M. Sato, T. Yamauchi and H. Taniguchi: Evaluation and Design of Function for Tracing Diffusion of Classified Information for File Operations with KVM, The Journal of Supercomputing, Vol.72, Issue 5, pp.1841–1861, (2, 2016). doi: 10.1007/s11227-016-1671-5
2. S. Fujii, M. Sato, T. Yamauchi and H. Taniguchi: Design of Function for Tracing Diffusion of Classified Information for IPC on KVM, Journal of Information Processing, Vol.24, No.5, pp.781–792, (9, 2016). doi: 10.2197/ipsjjip.24.781
3. H. Moriyama, T. Yamauchi, M. Sato and H. Taniguchi: Performance Improvement and Evaluation of Function for Tracing Diffusion of Classified Information on KVM, 2017 Fifth International Symposium on Computing and Networking (CANDAR), pp.463–468, (11, 2017). doi: 10.1109/CANDAR.2017.91
4. J. Hizver and T. C. Chiueh: Real-time deep virtual machine introspection and its applications, Proceedings of the 10th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp.3–14, (2014).
5. J. Shi, Y. Yang and C. Tang: Hardware assisted hypervisor introspection, SpringerPlus, 5(647) (2016)
6. D. Zhan, L. Ye, B. Fang, et al.: Checking virtual machine kernel control-flow integrity using a page-level dynamic tracing approach, Soft Comput 22, pp.7977–7987, (2018).
7. L. Jia, M. Zhu, B. Tu: T-VMI: Trusted Virtual Machine Introspection in Cloud Environments, Proceedings of 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp.478–487, (2017).
8. W. Enck et al.: TaintDroid: An information-Flow tracking system for realtime privacy monitoring on smartphones, ACM Transactions on Computer Systems, vol.32, no.2, Article 5, pp1–29, (2014).
9. Y. Ji et al.: RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS2017), pp.377–390, (2017).