國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

磁碟傷害範圍估測機制

A Storage-Layer Security Attack Damage Estimation Mechanism

研 究 生 ： 鄭又瑞

指 導 教 授 ： 吳育松 教授

中 華 民 國 103 年 8 月

磁碟傷害範圍估測機制

A Storage-Layer Security Attack Damage Estimation Mechanism

研 究 生 ： 鄭又瑞　　Student： Yu-Jui Cheng

指 導 教 授 ： 吳育松　　Advisor： Yu-Sung Wu

國立交通大學

資訊科學與工程研究所

碩士論文

A Thesis Submitted to Institute of Computer Science and Engineering College of Computer Science National Chiao Tung University in Partial Fulfillment of the Requirements for the Degree of Master in Computer and Information Science

August 2014

Hsinchu, Taiwan, Republic of China

中華民國一百零三年八月

磁碟傷害範圍估測機制

學生： 鄭又瑞　　指導教授： 吳育松 博士

國立交通大學資訊科學與工程研究所碩士班

摘　要

利用行為比對偵測惡意程式有很高的偵測率。然而觀測行為時，惡意程式仍持續對系統造成傷害，因此在判定惡意程式後，對其造成的傷害進行估測，可以協助管理者修復造成的系統傷害。

在半虛擬化的環境下，我們設計一套傷害範圍估測機制，藉由記錄在虛擬機中程式寫入的檔案路徑以及磁區位置，估測惡意程式造成的傷害範圍。我們修改 xen-blkback 攔截磁碟寫入的磁區位置，修改 Xen hypervisor 攔截系統呼叫，將兩者的 I/O 資訊合併進行傷害範圍估測。

關鍵字： 磁碟、傷害、估測

A Storage-Layer Security Attack Damage Estimation Mechanism

Student： Yu-Jui Cheng Advisor： Dr. Yu-Sung Wu

Institute of Computer Science and Engineering National Chiao Tung University

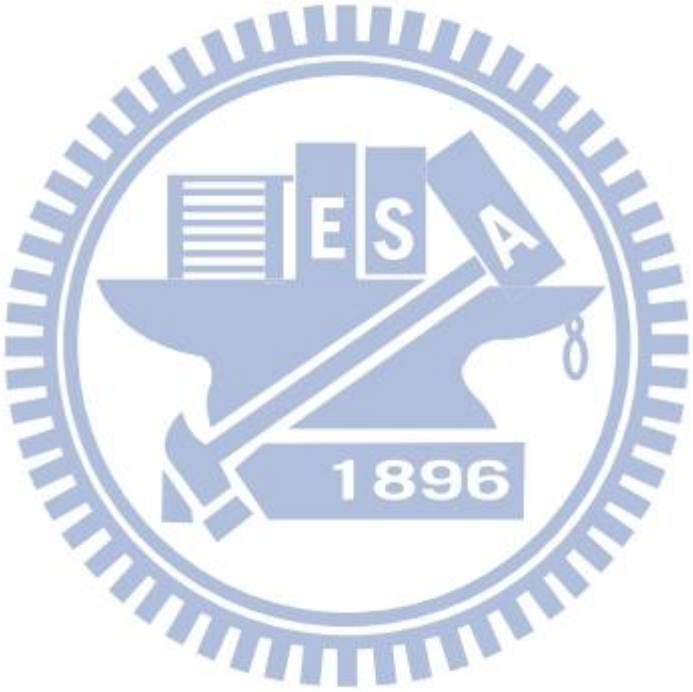
ABSTACT

Behavior matching is a malware detection method with high detection rate. However, during the time matching behaviors, the malware is continually making damage. Thus, estimating the damaged area the detected malware made can help administrator relieve the damage.

In paravirtualized environment, we design a storage-layer damage estimation mechanism. We estimate the damage that a malware made by using the disk I/O information from guest VM. We modify xen-blkback to intercept raw disk I/O information, and Xen hypervisor to intercept system calls. We combine raw disk information and system call information to estimate damaged area.

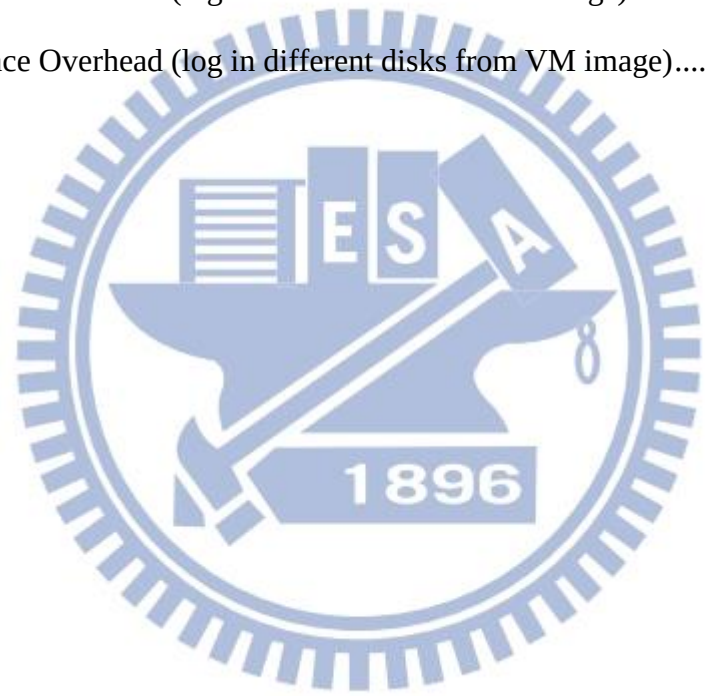
Keywords: *Storage, disk, damage, estimate*

Table of Contents



List of Figures

List of Tables



Chapter 1. Introduction

Behavior matching[1, 2] is a malware detection method with high detection rate. However, this method takes a period of time to verify whether a process is a malware or not. During this time, the malware is continually making damage. To relieve or eliminate the damage, it is necessary to find out the damaged area.

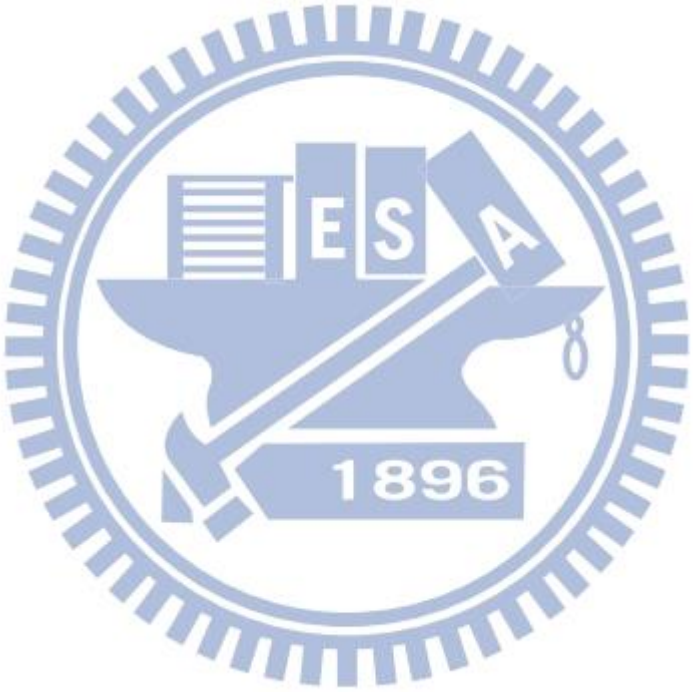
Security damage to a system can be separated to two types, memory-layer damage and storage-layer damage. Memory-layer damage can be eliminate after the machine reboot, but storage-layer damage cannot. Thus, we design a damage estimation mechanism to estimate storage-layer damage. The mechanism is composed of two parts, I/O interception and damage estimation. We intercept I/O requests and log them. When finding a malware, we use the logs to estimate damage of the malware.

In a virtualization datacenter, service providers are not expected to install any software in the virtual machine. Therefore, the I/O interception must be implemented outside of guest virtual machine. Xen[3] is a popular hypervisor that Amazon uses it[4], so we implement the damage estimation system in domain 0 on Xen. The damage estimation system composed of Written Area Logging Daemon which logs raw disk I/O and system call information, and Damage Estimator which estimates damaged area.

We modified xen-blkback driver to intercept raw disk I/O, and hypervisor to intercept system calls. In addition, we design a method to combine the information of raw disk I/O and system calls. Damage Estimator use the combined information to estimate damaged area. However, intercepting both raw disk I/O and system calls will cause more performance overhead. Thus, we also design sector mode and file mode damage estimation to roughly estimate damage with only one of raw disk I/O and system calls. The overhead of logging is lower if using rough damage estimation method, but the accuracy is also lower.

In Chapter 3, we introduce the design of our damage estimation mechanism, and in Chapter

4, we introduce the implementation. Chapter 5 shows the evaluation of the damage estimation system, and we discuss some problem we met. Chapter 6 is the Conclusion.



Chapter 2. Related Work

We estimate storage-layer damage is for administrator to relieve the damage. [5] finds malwares' behaviors and creates a targeted remediation program. However, the program must run in guest VM. The user may ignore it and the damage still exists. Our damage estimation mechanism is in host VM. We use xen-blkback to intercept raw I/O requests. Xen blktap[6] also provides an interface to intercept I/O requests. However, blktap does not support interception of block devices but only image files.

Windows Minifilter driver[7] also intercepts I/O requests. However, it can only run in Windows guest VM. Tools in guest VM may be avoided by malware. The Sleuth Kit[8] is a tool to analyze image file. We can get information about image file but no information about running processes. To estimate damage, we use ntfs-3g[9] to get file information in image file. Ntfs-3g is a library to access ntfs file system. It helps us to find out the sectors which file data resides in the disk.

We intercept I/O requests from guest VM, but if the requests are cached in guest VM, we cannot get the information before it flushes. However, [10] says that synchronous I/O in guest VM and asynchronous I/O in host VM performs good and stable. We can use this model so that we won't encounter the problem of cache in guest VM.

Chapter 3. Design of Damage Estimation Mechanism

A malware could damage a system by tampering with the data stored on persistent storage devices. For instance, a malware could modify system registries and replicate its executable binaries onto the file systems. We consider these as persistent damages because the effects will remain even after a reboot of the system. In fact, many of the damages are hard to identify or be removed as malware due to the clandestine nature of malware. Much prior work has focused on the detection of malware rather than on the identification and clean-up of the damages as caused by malware. To address the problem, we design a damage estimation system called Damage Estimator to estimate persistent damage caused by the execution of malware in a virtual machine.

3.1. Damage Estimation

The damage estimation mechanism is composed of three parts: read process' I/O information, decide damaged area, and output the estimated damaged area.

Before estimating the damaged area which a process made, Damage Estimator has to read the process' I/O information, so that it has sufficient information to decide damaged area. Thus, we design an I/O logging system to intercept and log I/O operations from guest VMs. When estimating damaged area, Damage Estimator reads the logs and extracts the I/O operations which we are interested in. We use process id and running time to distinguish the process in which we are interested from others because process id is unique at the same time.

I/O operations a guest VM did is like Figure 1. *Process A* and *Process B* makes some I/O operations, and the I/O operations may intermix. If we are interested in *Process A*, Damage Estimator only considers I/O operations of *Process A.* In Figure 2, I/O operations of *Process B* are ignored, and continual operations will be put into one set. We define a time threshold ε. If the time interval between two operations is less than ε, then we say they are continual.

There are three sets in Figure 2. The damage *Process A* did at time *t* is affected area in *Set Y.*

When ε is set to a small value, we can get more accurate damaged area. In this case, the estimated damaged area may not contain all I/O operations of *Process A*, but only a few I/O operations of *Process A* around time *t*. Sometimes, a process does untrusted actions, it is not completely untrusted. For example, a browser may open a malicious website and run the malicious scripts. The browser is not a malware, but it still makes some damage because of the malicious scripts. When ε is set to a large value, the estimated damaged area will contain all I/O operations of *Process A.* Therefore, the estimated damaged area will contain all I/O operations an untrusted process does.

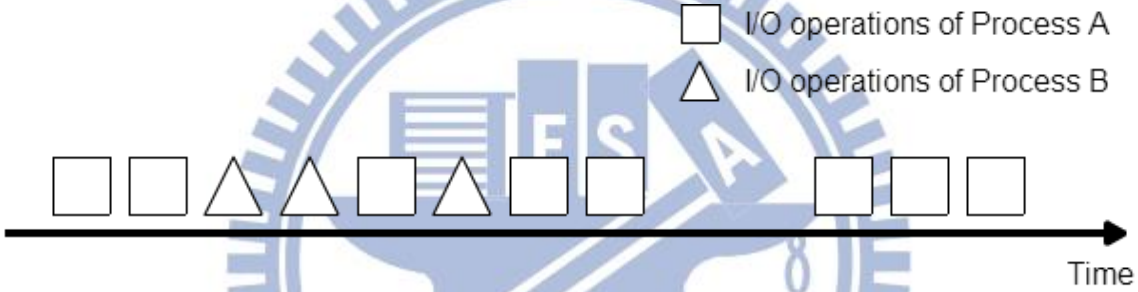


Figure 1 I/O operations of a guest VM

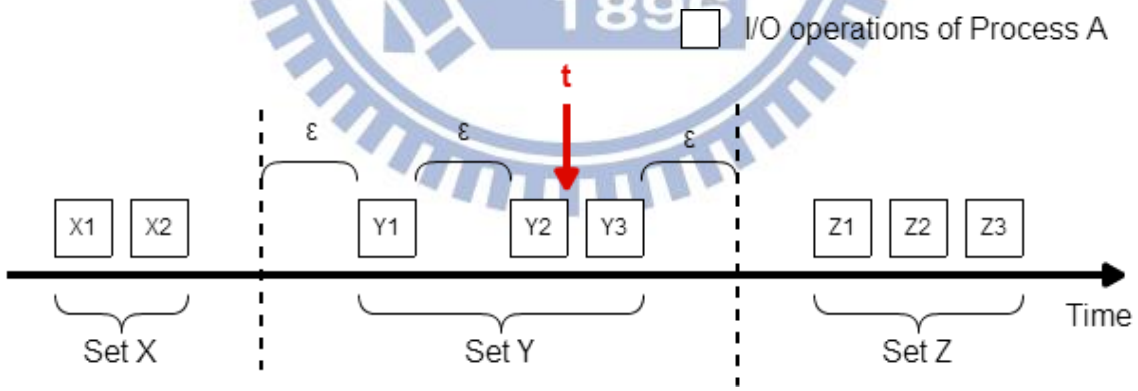


Figure 2 I/O Sets of Process A

Figure 3 shows the detail flow of damage estimation. First, we start Damage Estimator and input process id and time. Then, Damage Estimator will open log files and read them. Damage Estimator ignores logs with different process id from input. If the log's process id is the same as input, Damage Estimator will try to put it into a set. If no set exists, Damage Estimator will create a new one and put the log into it. If a set exists, Damage Estimator will check the elapsed

time from the last log in the set to the processing log. If the elapsed time is less than ε, the log will be put into the set. Otherwise, Damage Estimator creates a new set to replace the old one, and puts the log into the new set. Then, Damage Estimator reads the next log and processes it until the log time is greater than the input time. At the end, Damage Estimator output the logs in the set. If any error occurs, Damage Estimator will output it.

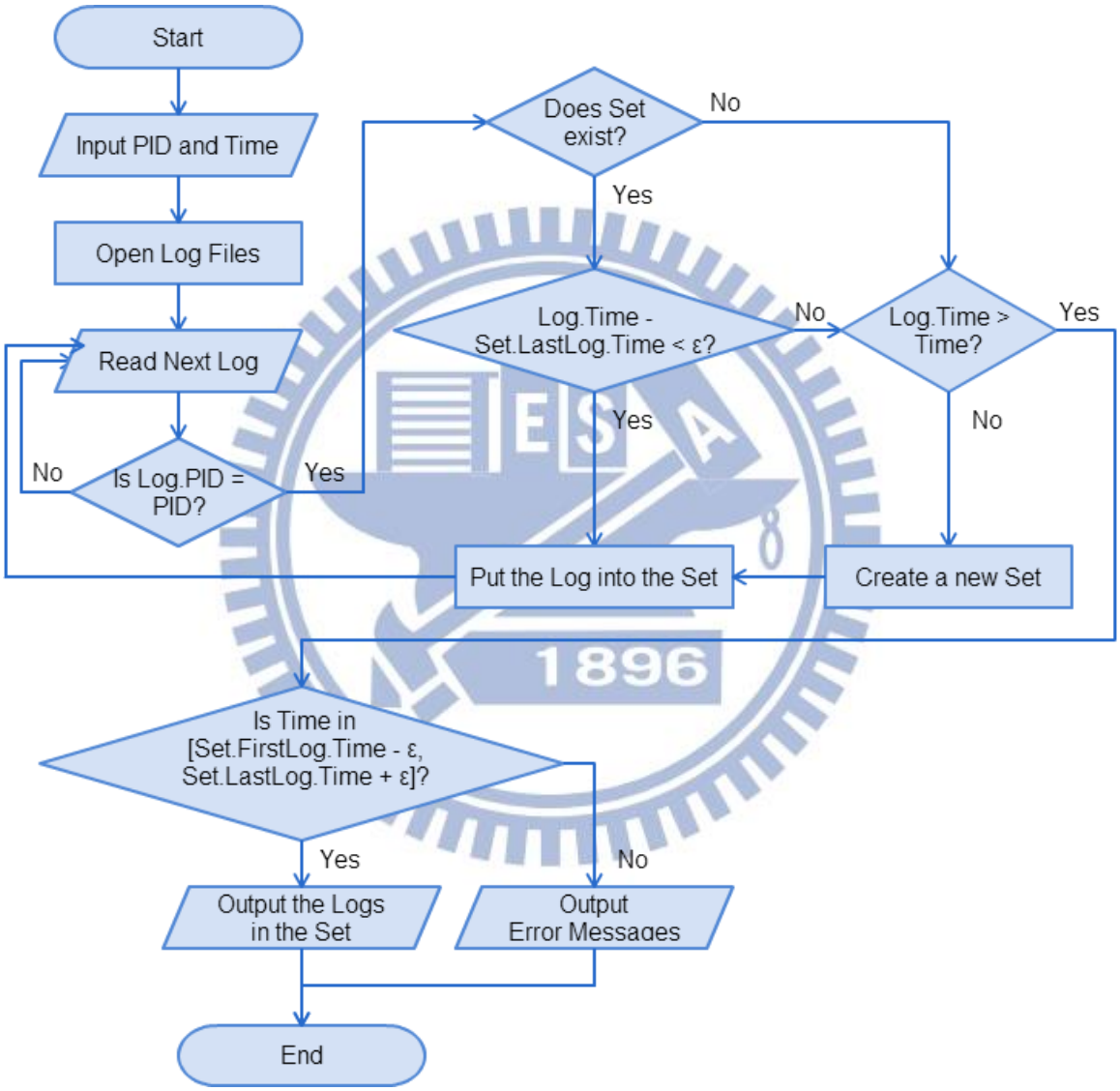


Figure 3 Damage Estimation Flow Chart

3.2. Architecture of Damage Estimator

Damage Estimator is built on paravirtualized(PV) environment. Paravirtualization provides higher performance than full-virtualization.

The damage estimation mechanism is based on the I/O information intercepted from guest VMs. Therefore, we design I/O logging system to intercept system calls and I/O requests to storage devices. I/O logging system is composed of three parts, system call interception, raw disk I/O interception, and Written Area Logging Daemon which gathers system calls and I/O requests.

3.2.1. Written Area Logging Daemon

Written Area Logging Daemon communicates with PV backend I/O Information Collector and System Call Information Collector to gather I/O information of system calls and raw disk I/O, and logs the information. PV backend I/O Information Collector intercepts raw disk I/O requests, and System Call Information Collector intercepts system calls from guest VMs. Damage Estimator needs I/O information to estimate damage. Written Area Logging Daemon is the part that collects all information Damage Estimator needs.

3.2.2. PV backend I/O Information Collector

In paravirtualized environment, PV frontend driver is installed in guest VM, and PV backend driver is installed in host VM. All I/O requests pass through PV frontend and backend driver. We choose PV backend driver to intercept I/O requests from guest VM. Because PV frontend driver is installed in guest VM, malwares may be able to avoid the interception. PV backend is out of guest VM, so the interception cannot be found and avoided in guest VM.

All raw disk I/O from guest VM can be intercepted in PV backend. However, raw disk I/O information is not detail enough. Thus, we intercept system calls to get more detail information.

3.2.3. System Call Information Collector

When a process wants to open a file and writes data to it, the process must request some system calls. System calls can be intercepted in hypervisor. There is a lot of types of system

calls, but we only consider open file and close file. We can get file path and process id from system calls of opening a file, and get the finish time of writing from system calls of closing a file. Though we can get system calls of writing a file, it does not contain addresses to write. We get the addresses to write from PV backend. Thus, we combine the information from PV backend and system call to generate process' I/O information to estimate damage.

3.3. Combine I/O Information of Raw Disk I/O and System Calls

Table 1 shows the information that we can get from raw disk I/O and system calls. We take time to combine them. Between file open and close, disk I/O which touches the file is correlative to the process. For Accuracy, we take overlapping area as damaged area.

Table 1 Information from raw disk I/O and system calls

Raw Disk I/O	System Calls
Written Sector Numbers Time	Opened/Closed File Path Process ID Time

Disk I/O uses sector as a unit. However, a file may not align to sectors. Thus, Figure 4 shows four cases when a process writes data to a file. Assume the left border of written sector is x, and the right border of written sector is y. The left border of file is p, and the right border of file is q. In Figure 4, we can find that the overlapping area is [max(x, p), min(y, q)].

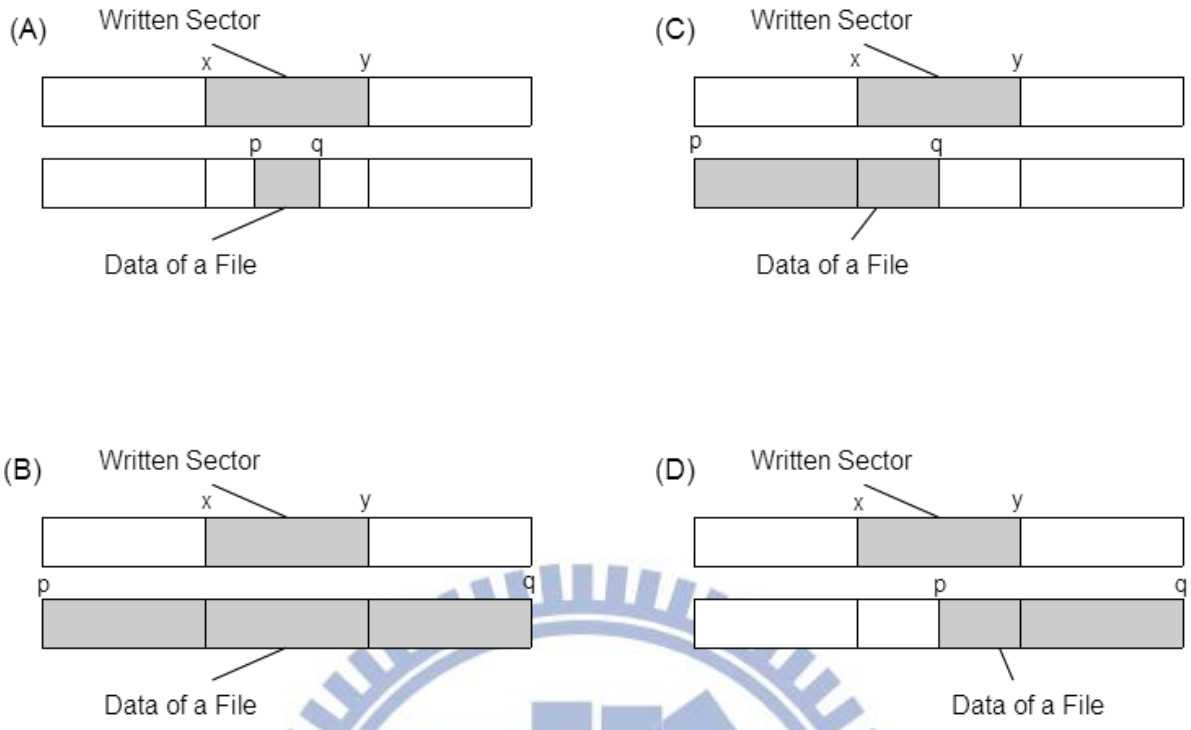


Figure 4 four cases when writing data to a file

3.4. **Rough Damage Estimation**

I/O interception makes the performance lower. For higher performance, we can only log either raw disk I/O or system calls. However, the damage estimation mechanism needs both raw disk I/O and system call information. Thus, we design sector mode and file mode to roughly estimate damaged area.

3.4.1. Sector Mode

If we have only raw disk I/O information, we can only use time to determine damaged area. Thus, the idea of sector mode damage estimation is finding out the damaged area during the given time interval. Figure 5 shows how sector mode estimation works. At beginning, input two time into Damage Estimator. Then, Damage Estimator read logs one by one. If the time of the log is in the interval Time1 to Time2, output the log. Otherwise, ignore the log and read next one.

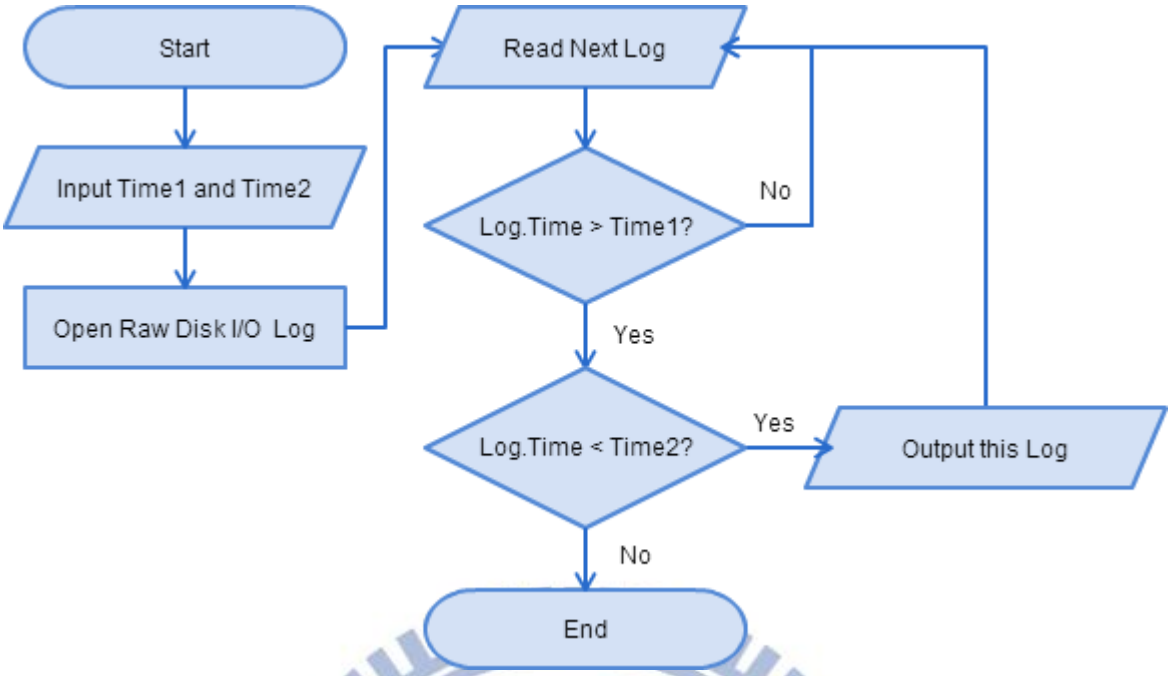


Figure 5 Sector Mode Estimation Flow Chart

3.4.2. File Mode

If we have only system call information, we can use process id and time to determine damaged area. The idea of file mode damage estimation is finding out the files which the given process is opening at the given time. Figure 6 shows how file mode damage estimation works.

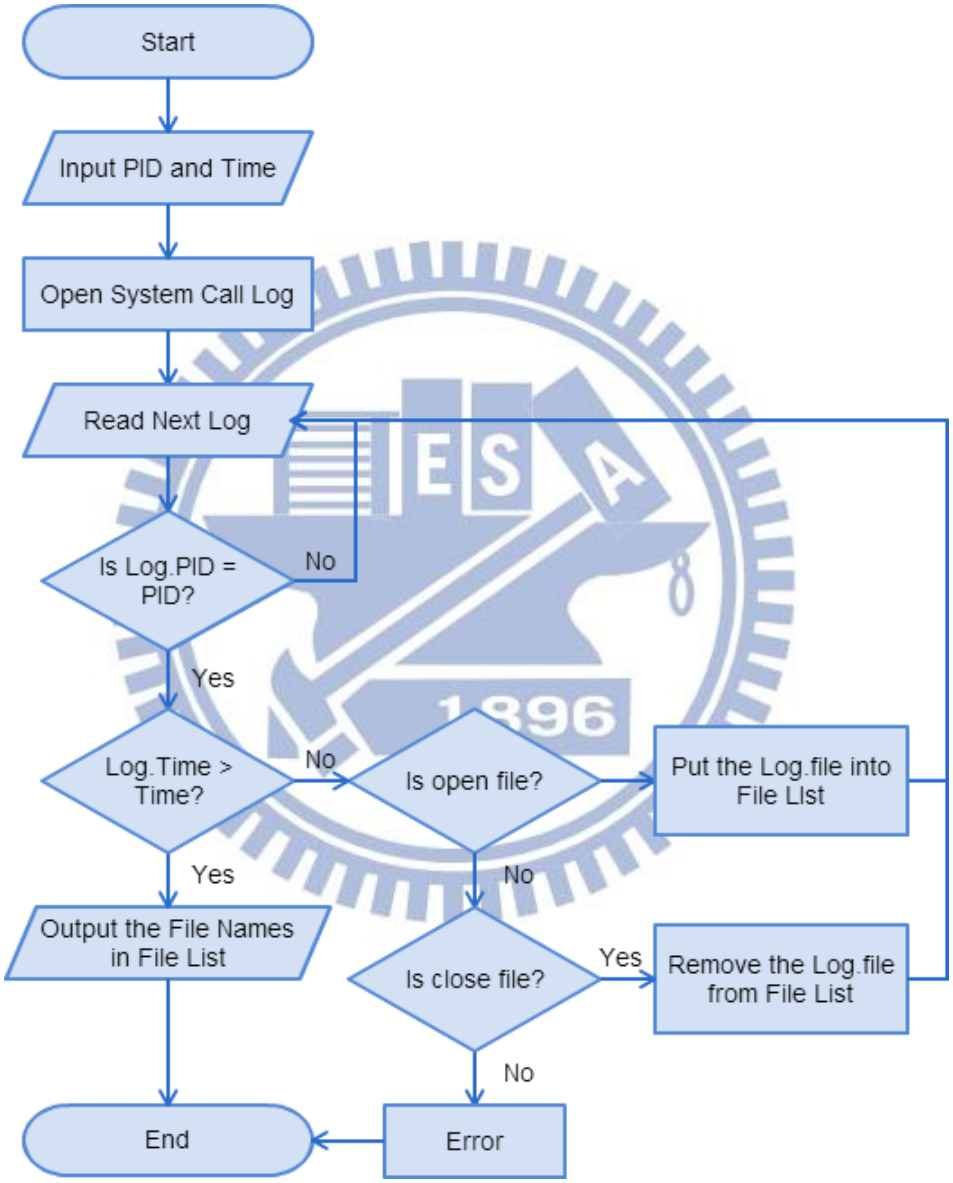
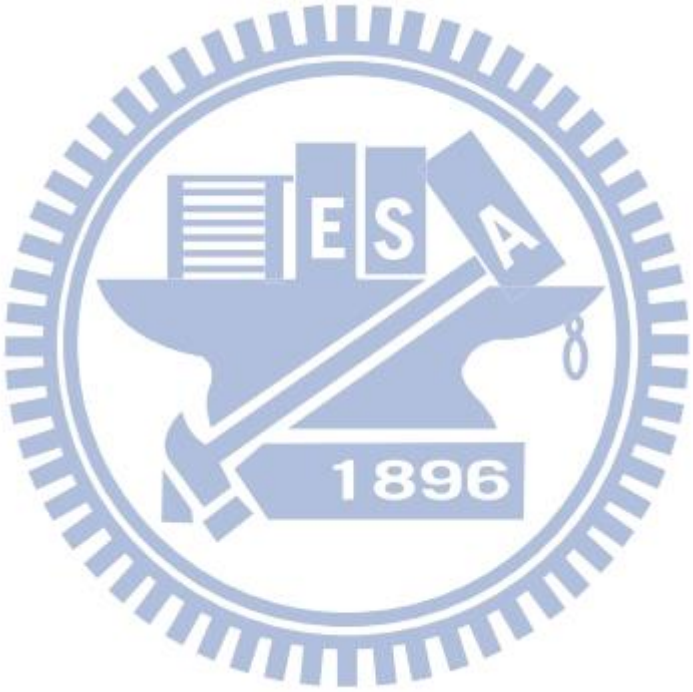


Figure 6 File Mode Estimation Flow Chart

First, open the log and read log one by one. Then, Damage Estimator considers only the logs which process id is equal to the given process id. Damage Estimator maintains a list. If the system call of the log is opening a file, Damage Estimator puts this log into the list. If the system

call of the log is closing a file, Damage Estimator removes the corresponding log in the list. At the end, Damage Estimator stops processing and output logs in the list after reading a log which is over the given time.



Chapter 4. Implementation

We implement Damage Estimator on Xen, and we use Windows as guest OS. Figure 7 is the architecture of whole Damage Estimation System. Written Area Logging Daemon runs in background and logs all write requests in domain 0. When Damage Estimator need the logs to estimate damaged area, it reads the logs and extracts needed information.

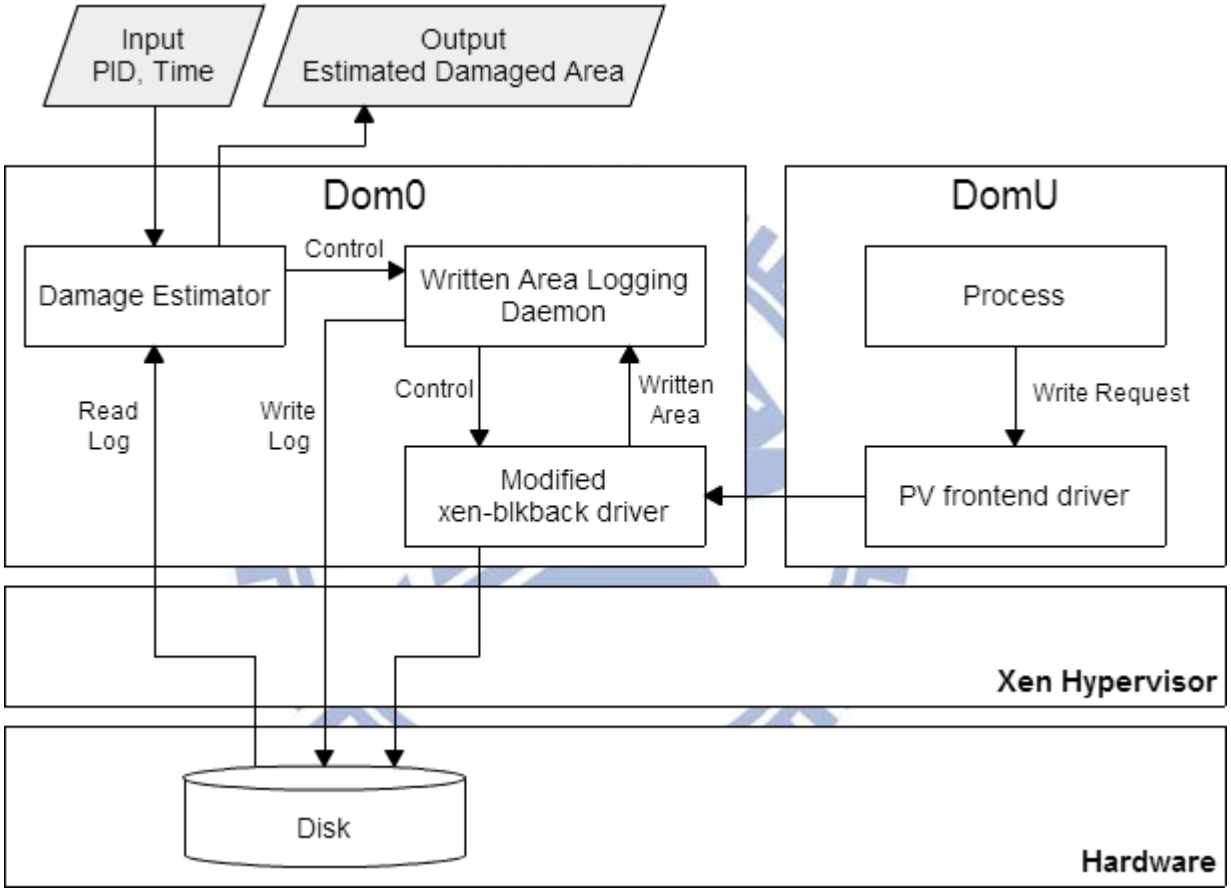


Figure 7 Architecture

4.1. Xen-blkback driver

The PV backend driver of Xen is xen-blkback. We modify dispatch_rw_block_io function in xen-blkback to intercept raw disk I/O and use netlink library to pass the I/O information to Written Area Logging Daemon. All read and write requests pass through dispatch_rw_block_io function. It receives the requests from upper system. After transmuting the requests to a proper

structure, this function pass the structure to the underlying storage. Thus, we can intercept all raw disk I/O requests here.

4.2. Interception of System Calls

System Call Information Collector uses CPUID based method mentioned in [11] to intercept system calls. Damage Estimator needs only NtCreateFile, NtOpenFile, and NtClose system calls. NtCreateFile and NtOpenFile are for opening file. When the file does not exist, NtCreateFile creates a new one and open it, and NtOpenFile returns an exception. If the file exists, both NtCreateFile and NtOpenFile open the file and return the file handle. NtClose uses the file handle to close the file.

Written Area Logging Daemon receives decoded system calls from System Call Information Collector. The decoded system calls contain process id and file path. However, Damage Estimator needs the area of the file to combine I/O information with raw disk I/O. Thus, we use ntfs-3g library to parse NTFS file system in the image file. After parsing, Written Area Logging Daemon logs operation, pid, path, and area of the file.

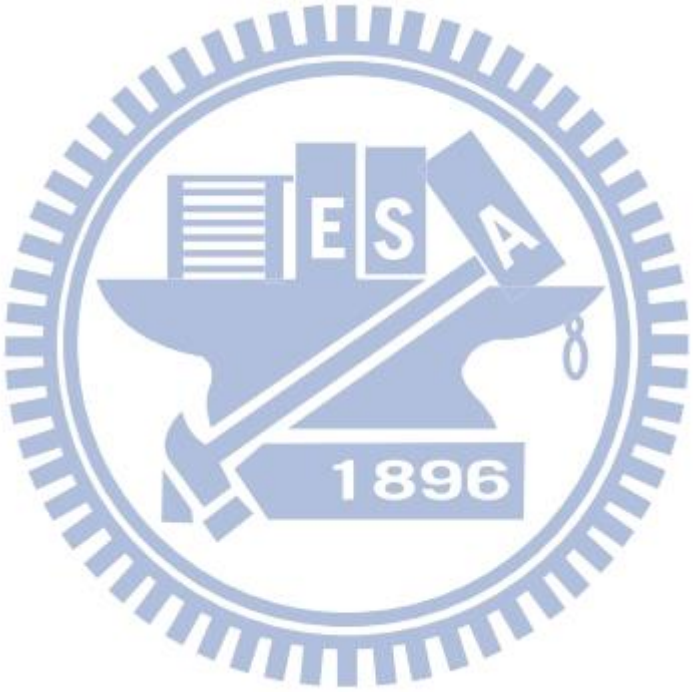
4.3. Written Area Logging Daemon

Written Area Logging Daemon communicates with xen-blkback driver and system call Information Collector. When an I/O request is sent from guest VM, xen-blkback and system call Information collector will trigger an event and send the I/O information to Written Area Logging Daemon to log it. Written Area Logging Daemon can also log only one of raw disk I/O and system call information. This will provide higher performance, but Damage Estimator can use only the method mentioned in section 3.4 to roughly estimate damage.

4.4. Damage Estimator

Damage Estimator read logs and combines raw disk I/O information and system call I/O

information. Then, Damage Estimator use the method mentioned in section 3.1 to estimate the damaged area. If there is only one of raw disk I/O and system call log, Damage Estimator will use the method mentioned in section 3.4 to roughly estimate damaged area.



Chapter 5. Evaluation

In this section, we perform a series experiments to evaluate Damage Estimator. In section 5.1, we describe the testing environment. In section 5.2, we compare the performance of full virtualization with paravirtualization. In section 0, we show the functionality of Damage Estimator. In section 5.4, we show the performance overhead when using our damage estimation system. In section 5.5, we have some discussion of the damage estimation mechanism.

5.1. Test Environment

We perform the experiments on a server with Dual Intel Xeon E5520 16-core processor, 16GB RAM, and 2TB hard disk. The virtualization environment is Xen 4.2.1 on CentOS with Linux kernel 3.10.35, and Windows 7 for guest VM. In guest VM, there is two types of disk, qemu disk and PV disk. Qemu disk is full virtualized storage device, and PV disk is paravirtualized storage device. We use fio[12] to evaluate disk I/O performance.

5.2. Disk I/O Performance of Full Virtualization and Paravirtualization

In this section, we use fio to evaluate disk I/O performance of full virtualization and paravirtualization. Our damage estimation mechanism is only related to disk writing. Thus, we perform an experiment to evaluate disk writing bandwidth. We compare disk sequential writing bandwidth with different size of I/O unit. Table 2 shows disk writing bandwidth with 512Bytes, 4KB, 32KB, and 512KB I/O unit. When fio takes 512Bytes as I/O unit, PV disk gives worse bandwidth. In the other cases, PV disk performs better than qemu disk does.

Table 2 Sequential Writing bandwidth

	512B	4K	32K	512K
Domain 0	20.0	426.1	3289.1	41817.0
Qemu Disk	39.3	321.5	2569.4	9655.3
PV Disk	19.4	422.0	3284.5	41097.0

(KB/s)

In Figure 8, we show the proportion of writing bandwidth of qemu disk and PV disk to host VM. It is obvious that the writing bandwidth of PV disk is close to the writing bandwidth of domain 0. The writing bandwidth of qemu disk is about 200% in 512 Bytes I/O unit case, and 25% in 512 KB I/O unit case. In 4KB and 32KB cases, the writing bandwidth of qemu disk are about 75%. We think the difference in these cases is because of the I/O buffer of qemu disk. If the I/O unit is less than the buffer size, qemu disk can finish more I/O requests when flushing the buffer. In the contrast, if the I/O unit is more than the buffer size, one I/O request may be split into two or more parts. Therefore, the performance of qemu disk is worse in 512KB case. In the other hand, PV disk works like domain 0 does. The overhead of PV disk is the communication of PV frontend and backend driver, so the performance of PV disk is close to domain 0. Thus, we implement Damage Estimator on paravirtualized environment.

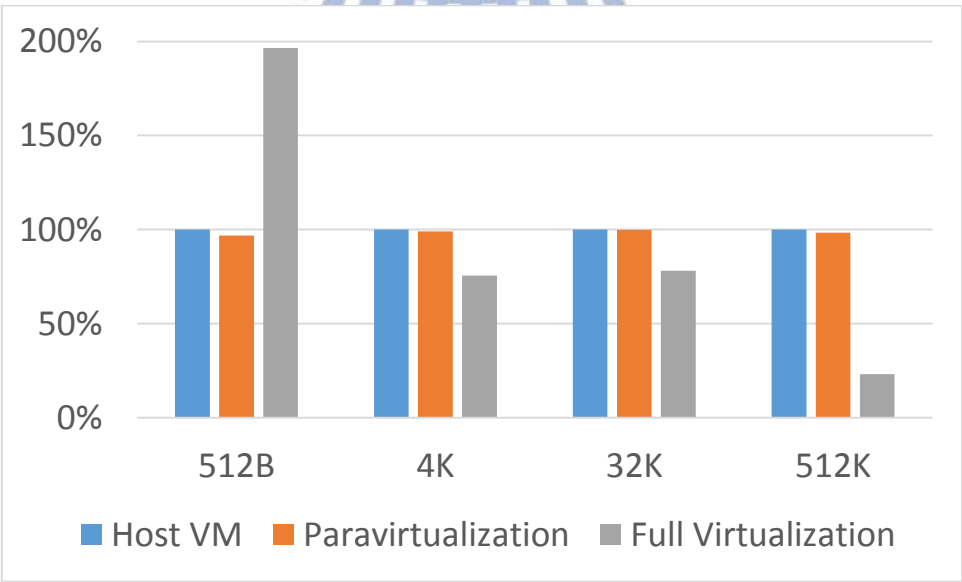


Figure 8 Proportion of writing bandwidth to host VM

5.3. Functionality of Damage Estimator

To evaluate the functionality of Damage Estimator, we design a test program which writes assigned data and output the running time and its process ID. Written Area Logging Daemon logs the I/O information, and Damage Estimator estimates damaged area in three modes. Section 5.3.1 is the result of sector mode estimation, and section 5.3.2 is the result of file mode estimation. In section 0, we combined raw disk I/O and system calls to estimate damage, and show the result.

5.3.1. Sector mode estimation

Sector mode estimation can only use time to decide damaged area. We use the test program to write 512 Bytes of data. However, Damage Estimator gives a large area of damage. We run the program at timestamp 1408472852, but we intercept it at timestamp 1408472855. Table 3 is the result of sector mode damage estimation. The write request delays 3 seconds. In addition, the test program write only 512 Bytes, but the intercepted request which contains the data the test program write is 4096 Bytes. This is because the sector size of domain 0 is 4096 Bytes. Though the sector size of guest VM is 512 Bytes, the minimum size PV backend driver to write is 4096 Bytes.

Besides, we found that only use time to estimate damage is not a good way. There may be a lot of I/O requests in a second, but only one request we are interested in. This method is easy, but not accurate. Only use raw disk I/O information to estimate damage cannot give a satisfied result.

Table 3 Result of Sector Mode Damage Estimation

No.	Timestamp	Offset	Length
1	1408472853	223617024	8192
2	1408472855	3327651840	4096
3	1408472855	106012672	4096
4	1408472855	3235426304	20480
5	1408472855	3331166208	4096
6	1408472855	3191869440	4096
7	1408472855	3345383424	4096
8	1408472855	3389915136	4096
9	**1408472855**	**17162031104**	**4096**
10	1408472855	17164312576	8192
11	1408472855	17176997888	4096

5.3.2. File mode estimation

Damage Estimator in file mode estimation takes the assigned time and process ID to find out files the process opened at the assigned time. Table 4 is the result of file mode damage estimation, and written area stores the data the test program write. Table 4 (A) is the case of writing data to a 512 Bytes file. We do not consider the metadata of the file, so the result contains only the 512 Bytes the test program writes.

Sometimes the area of a file may not be continually. We use the test program to write 32 KB of data into a file. Table 4 (B) shows the result. In Table 4 (B), we can find that the data is composed of three different area. If we read the area of the VM image file, we can find the assigned data in it.

Table 4 Result of File Mode Damage Estimation

(A)

No.	Process ID	Path	Written Area
1	**1444**	**c:\written_data**	**[17162034448, 17162034959]**

(B)

No.	Process ID	Path	Written Area
1	**1788**	**c:\written_data**	**[442351616, 442355712], [396996608, 397008896], [676003840, 676020224]**

5.3.3. Damage Estimation with combined I/O requests

Damage Estimator combines raw disk I/O and system call information to estimate damage. We create a 32 KB file and write 1024 Bytes of data at offset 5000. Table 5 shows the result of damage estimation with combined I/O requests. Table 5 (A) shows the sector mode estimation of this case. The data we write is at the 5th row. Table 5 (B) is the area of the file. Table 5 (C) is the result with combined I/O information.

The 1024 Bytes of data written by the test program is in the 4096 Bytes estimated area. Damage Estimator combines the results in Table 5 (A) and (B). Then, we get the overlapping area [396996608, 397000704]. Comparing this area with the area in Table 5 (A) and (B), we find that using combined I/O information will get more precise area. Sector Mode Estimation gives too large area, and File Mode Estimation gives the whole area of the file but we only modify a little.

Table 5 Result of Damage Estimation with combined I/O requests

(A)

No.	Timestamp	Offset	Length
1	1408973426	3231072256	20480
2	1408973426	3191865344	4096
3	1408973426	3191861248	4096
4	1408973427	117379072	8192
5	**1408973429**	**396996608**	**4096**

(B)

No.	Process ID	Path	Written Area
1	**2200**	**c:\written_data**	**[442351616, 442355712], [396996608, 397008896], [676003840, 676020224]**

(C)

Process ID	Timestamp	Written Area
2200	**1408973429**	**[396996608, 397000704]**

5.4. **Performance Overhead**

We show the performance overhead of our damage estimation system in this section. Written Area Logging Daemon intercepts all I/O requests. Thus, disk I/O performance is mainly affected by Written Area Logging Daemon. Besides, if the logs and the image file are stored in different storage device, disk I/O performance will be better. Table 6 is the performance overhead of Written Logging Daemon which logs in the same disk as VM image file. Table 7 is the performance overhead which the logs are stored in different disks from VM image file.

Table 6 Performance Overhead (log in the same disk as VM image)

	512B	4K	32K	512K
Original PV Guest VM	19.4	422.0	3284.5	41097.0
Raw Disk I/O + System Calls	7.4	107.7	856.1	6560.7
Raw Disk I/O Only	9.6	197.4	998.7	8424.6
System Calls Only	13.5	179.3	1010.3	7260.3

(KB/s)

Table 7 Performance Overhead (log in different disks from VM image)

	512B	4K	32K	512K
Original PV Guest VM	19.4	422.0	3284.5	41097.0
Raw Disk I/O + System Calls	13.8	128.7	1068.5	7353.9
Raw Disk I/O Only	19.3	372.9	3170.6	22482.0
System Calls Only	14.8	134.1	1093.7	8921.4

(KB/s)

We can find that logging in different disks from VM image file performs better. We take the writing of original PV guest VM as the baseline, Figure 9 and Figure 10 is the proportion of writing bandwidth to original PV guest VM. It is obvious that logs in different disk makes raw disk I/O logging perform better, but system call logging do not. Raw disk I/O logging in different disk from VM image file is close to original PV guest VM.

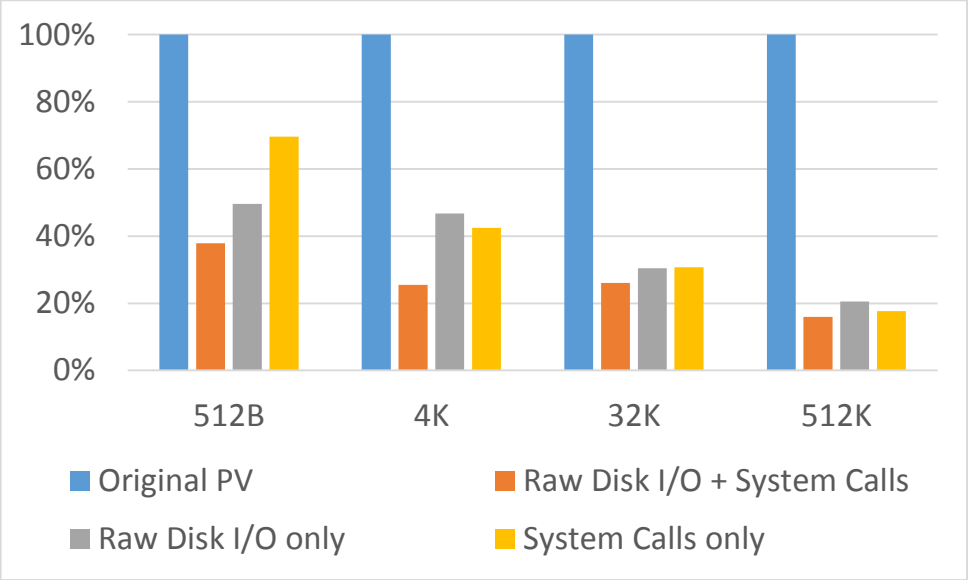


Figure 9 Proportion of writing bandwidth to original PV. Logs and image file are in the same storage device

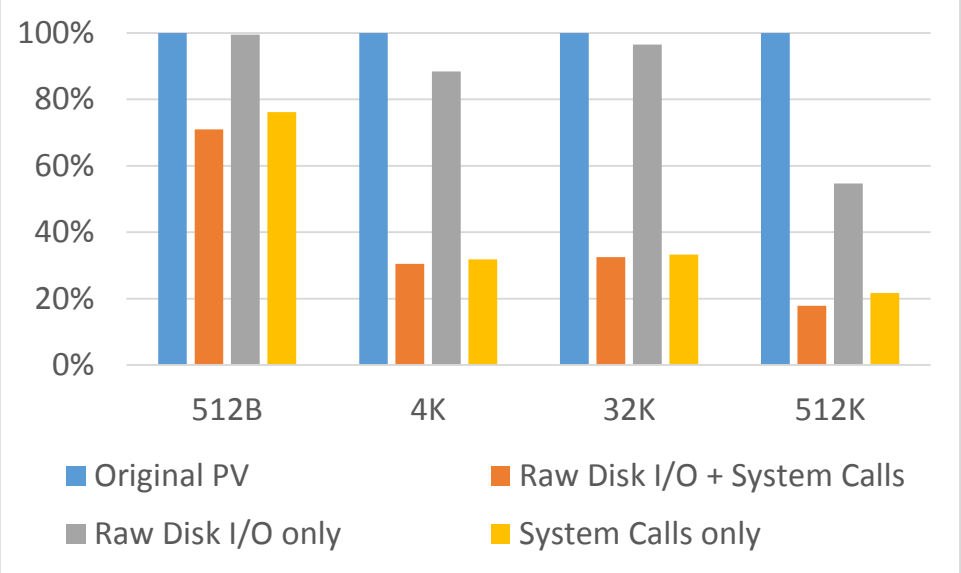


Figure 10 Proportion of writing bandwidth to original PV. Logs and image file are in different storage device.

5.5. **Discussion**

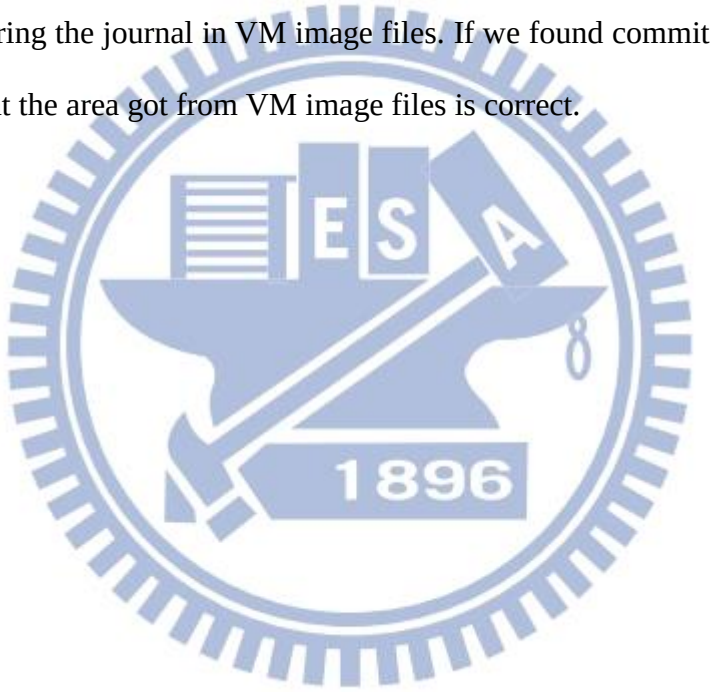
5.5.1. Time Synchronization of Guest VM and Host VM:

Our damage estimation mechanism is built on host VM. Thus, time synchronization of guest VM and host VM is a problem because the time in logs may not equal to the input time for Damage Estimator. Written Area Logging Daemon uses the time in host VM, but users use

the time in in guest VMs. The best way to solve the problem is to synchronize the time in guest VM and host VM.

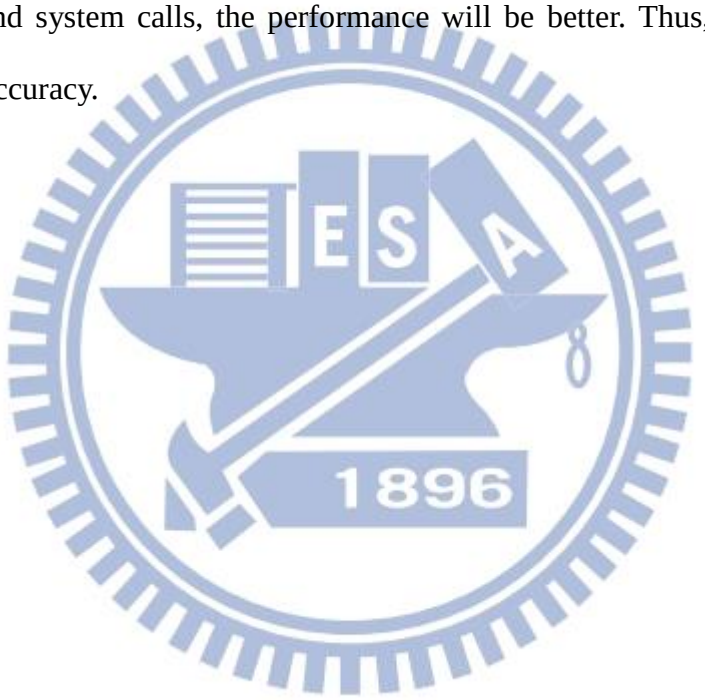
5.5.2. File Synchronization of Guest VM and Host VM:

We intercept system calls to get some file information. When we get a file path, we read the VM image file to get the area which the file used. However, when a file modification occurs, the write operation may not take an effect to VM image files right away. The area got from image files may not be correct because of the cache which is handled by guest VM. The possible solution is considering the journal in VM image files. If we found commit log of journal, then we can confirm that the area got from VM image files is correct.



Chapter 6. Conclusion

We design a damage estimation mechanism by using the logs of raw disk I/O and system calls. We modify xen-blkback driver and hypervisor to intercept I/O information. Written Area Logging Daemon logs the information and Damage Estimator uses the logs to estimate damaged area. Damage Estimator combines raw disk I/O and system call information to estimate damage. If Damage Estimator uses only one of raw disk I/O and system calls to estimate damage, the accuracy will be lower. If Written Area Logging Daemon logs only one of raw disk I/O and system calls, the performance will be better. Thus, it is a trade-off of performance and accuracy.



Reference

[1] L. Martignoni, R. Paleari, and D. Bruschi, "A framework for behavior-based malware analysis in the cloud," in *Information Systems Security*, ed: Springer, 2009, pp. 178-192.

[2] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *ACM India Software Engineering Conference*, 2008, pp. 5-14.

[3] *The Xen Project*. Available: http://www.xenproject.org/

[4] *Citrix and Amazon Web Services (AWS)*. Available: http://www.citrix.com/global-partners/amazon-web-services.html

[5] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. T. Giffin*, et al.*, "Automatic Generation of Remediation Procedures for Malware Infections," in *USENIX Security Symposium*, 2010, pp. 419-434.

[6] *Xen - Blktap*. Available: http://wiki.xen.org/wiki/Blktap

[7] *File System Minifilter Drivers*. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff540402(v=vs.85).aspx

[8] *The Sleuth Kit*. Available: http://www.sleuthkit.org/

[9] *NTFS-3G + Ntfsprogs*. Available: http://www.tuxera.com/community/ntfs-3g-download/

[10] D. Li, X. Liao, H. Jin, B. Zhou, and Q. Zhang, "A new disk i/o model of virtualized cloud environment," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 24, pp. 1129-1138, 2013.

[11] Y.-S. Wu, P.-K. Sun, C.-C. Huang, S.-J. Lu, S.-F. Lai, and Y.-Y. Chen, "EagleEye: Towards mandatory security monitoring in virtualized datacenter environment," in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1-12.

[12] *fio*. Available: http://freecode.com/projects/fio