# Vis: Virtualization Enhanced Live Forensics Acquisition for Native System


## Abstract

Live forensic is becoming one significant part in modern digital investigation. It is effective in obtaining criminal evidence which only exists in memory. Unfortunately, current efforts either fail to provide accurate acquisition of native system state at the given time point or require suspending the machine and altering the execution environment drastically.

To address this issue, we propose Vis, a light-weight virtualization approach to provide accurate retrieving of native system state while preserving the execution of target system. Vis is built on two key technologies. The first one - Virtual-Snapshot - ensures the accuracy of the dumped system state without suspending the target system. The second one - Late-Virtualization - builds the required virtualization environment by encapsulating the native system into a single virtual machine after the OS finishes booting without environment impact upon the target system execution.

Our experimental results indicate that Vis is capable of reliably retrieving an accurate system image. Besides, Vis accomplishes live acquisition within 97.09∼105.86 seconds, which proves Vis is practical when comparing with hours needed by previous remote live acquisition tools and even days needed in static acquisition. In average, Vis introduces only 9.62% performance overhead to the target system.


## 1 Introduction

A typical computer forensics scenario has three steps: Acquisition, Analyzing and Reporting [38]. Focusing on the acquisition and analyzing stage, computer forensics proposes two key challenges: how to obtain the complete system state and how to analyze the retrieved image effectively [31]. The former problem is more important, since missing evidence leads to an incomplete or wrong investigation result, even with an ideal analyzing technology.

Most static forensic tools concentrate on the data in file systems because criminal evidence is regarded to be stored on permanent I/O device only [10]. In order to get the critical information effectively, static forensic examiners require to clone disks offline and then search the images for possible evidences. Traditional static forensic tools, like Encase [22] and FTK [3], are able to dig out massive volumes of installed programs, deleted files, email contents and browsing history from the images. Nevertheless, due to the significant growth in memory capacity, a wealth of evidence can only exist in volatile memory, which is totally beyond the acquisition scope of static forensic tools. Static forensic is also limited in practice with enterprise servers. With a requirement of 24/7 availability, these servers cannot bear the economic loss brought in by offline forensic operations. In the meantime, aiming at improving both the capacity and robustness, a distributed storage manner adopted in server extends the retrieving time from hours to days, even months [5].

One solution to address this issue is to perform *live forensic* on a target system. Revealed by its name, live forensic focuses on obtaining and analyzing the target system while keeping it in a running state [5]. Transcending static analysis approach, it extends the examiners's information gathering range to the volatile data, such as process information [8], process list [23], kernel objects [18] and raw memory content [6, 33]. From the architecture perspective, previous software live acquisition solutions can be divided into two categories. The first one is *Virtualization Introspection*, which means the target system is wrapped in a Virtual Machine (VM) while the acquisition module exists in hypervisor like Xen [16]. VIX tools [23], Ruo's work [6], Srinivas's work [26] and BodySnatcher [34] all belong to this type. The second one is Non-Virtualization Introspection, named *In-OS State Fetching* in this paper because no previous work supposes any native out-of-OS live acquisition ap-

proach. It is designed to obtain indicated volatile system state and data with a minimal environment impact. In Iain's work [35] there lists several practical tools for different scenarios, e.g Win32dd [30], KnTTools [20] and Fport [29]. Memoryze [27] is another popular user process forensic tool in this type.

While owning the ability of unearthing tremendous volume of volatile data, live analysis also faces significant challenges and risks. The first challenge is previous virtualization based live acquisition methods, especially those required to load hypervisor prior than Operating System (OS), alter the system environment significantly. When employing this method on a non-virtualized host, the forensic examiners more or less change the system running environment. In the extreme case, rebooting even reinstalling the whole system is required, causing a great loss of information from volatile memory. The second challenge is raised from the fact that the system is not static [5]. The continuous changing files and processes make those previous in-OS live acquisition methods unable to guarantee the accuracy of the retrieved *Machine State*, which includes system running environment and volatile data, at the given time point unless suspending the machine [31]. Figure 1 shows the accuracy evaluation of different common studied live forensic tools. It proves the inaccuracy of some existing in-OS live forensic methods. In Section 4, we will analyze this result in detail. Moreover, it is noteworthy that dumping an accurate machine state is still difficult by manipulating all page tables for in-OS live acquisition tools, because possible existence of hiding processes make it extremely difficult to actively trace all working page tables.

In addition, it is possible that a system is configured to detect a live analysis attempt (e.g., suppose that a suspect installed a hardware component that requires a user to hit the escape key within 5 seconds to maintain system configurations) and to delete incriminating data [23] or even to destroy the whole physical machine if detected. In these conditions, stealthy live acquisition approaches should be employed. Unfortunately, even for virtualization based live acquisition, existing reliable volatile memory acquisition methods still need to suspend the whole machine for a period of time, e.g., VMWare Workstation [37]. Meanwhile, the acquisition task takes longer when transmitting data over cables. It is reported that BodySnatcher requires suspending the target system for 45 minutes to accomplish a complete 128MB RAM acquisition over 115kbps serial I/O cable [34].

Virtualization has three essential characteristics: isolation [14, 19, 36], introspection [6, 25] and performance [4]. Inspired by the recent resurgence of system virtualization and its application to commodity processors, we adopt virtualization to address these problems.

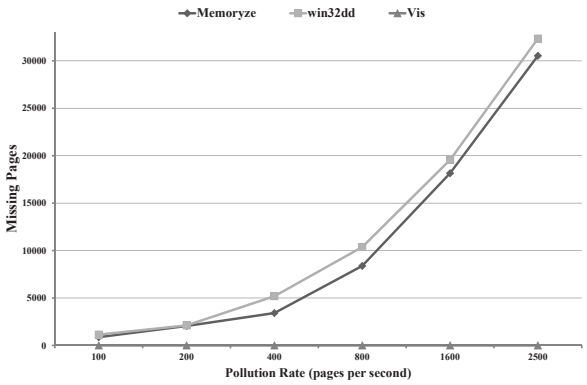


Figure 1: **Accuracy evaluation.** Different page pollution rate is tested for 2GB memory dumping in each test. Missing Pages means the number of the obtained pages containing polluted content.

We propose *Vis*, a light-weight virtualization approach, for commercial OSes to obtain accurate retrieved system state without environment impact while the target system keeps running.

To solve the first challenge in live acquisition, we introduce *Late-Virtualization* technique to ensure no environment impact is incurred during Vis lifetime. In order to be more applicable in real scenarios, Late-Virtualization also needs to build the virtual introspection environment after the OS is started up as well as keep hypervisor functioning without suspending the target system. To achieve these goals, it is necessary for Late-Virtualization to record and keep the original machine control state. Besides, Vis employs *Virtual-Snapshot* technique to provide accurate acquisition which solves the second challenge. Virtual-Snapshot leverages this virtual introspection environment to satisfy accurate system state retrieving. This is done by actively removing and granting write access of certain address space range on the additional level of address translation.

To the best of our knowledge, Vis is the first tool capable of providing accurate live acquisition for native system without suspending the target. To validate our approach, we have implemented a proof-of-concept prototype and conducted a series of evaluation. As shown in Figure 1, even under the high pollution rate for a period of time, Vis can still ensure the result accuracy while preserving the target system execution. Moreover, the performance evaluation result demonstrates that Vis is able to retrieve an accurate system image in 97.09$\sim$154.57 seconds comparing with a range of 17$\sim$76 seconds for existing live acquisition tools, while it incurs 9.62% performance overhead to existing applications. These results prove that Vis is practical in real world scenarios.

The rest of the paper is organized as follows. Sec-

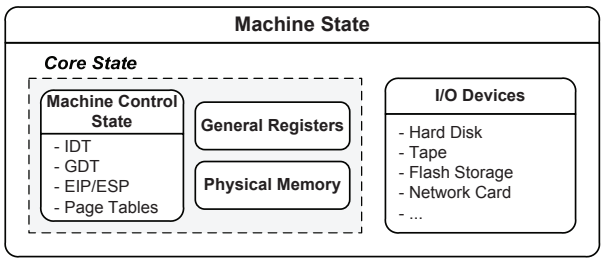


Figure 2: **Machine State Scope.**

tion 2 presents Vis's design model and assumptions. Section 3 provides the implementation details and related discussions, while Section 4 evaluates Vis through experiments. We survey related work in Section 5, then illustrate the future work and conclude in Section 6.

## 2 Design

For non-virtualized hosts, Vis ensures the accuracy of the retrieved machine state at the given time point. In this section, we begin with an illustration of how we define machine state, and then present the assumption we made in this paper. At last, we describe Vis's design and corresponding key technologies in detail.

### 2.1 Machine Context

The highest priority action of live acquisition is to distinguish which kind of information is needed to reconstruct the scene of crime. Generally, it needs to obtain the running environment information as well as both volatile and persistent state of a physical host, all of which constitute the machine state. A well defined machine state scope determines to which extent the target's information should be available to forensic practitioners. In our study, machine state contains the following three items: *Machine Control State (MCS)*, General registers and main memory, as well as Disk and other I/O device. MCS includes all host control related data, e.g., Interrupt Descriptor Table (IDT), Global Descriptor Table (GDT) and page tables. With these information, MCS is able to describe how the target system is executing currently. General registers and main memory, which are used to store the volatile system data, reveals what the target system is executing now. At last, acting as data warehouse, Disk and other I/O device supply tremendous volume of auxiliary information to construct the complete scene.

Figure 2 shows their relation hierarchy. Similar to storage hierarchy, various registers hold architecture information, describing the functionality of specific main memory content as well as how the main memory is used. Then, main memory holds file information in form of file descriptors stored in process structure. As a result, obtaining these registers and main memory content enables forensic examiners to form a better understanding of what the target system has done. It worths noting that the content in transparent memory devices (e.g., Cache and Translation Look-aside Buffer (TLB)) is unnecessary to be obtained because losing their content has no effect to the evidence integrity. In this paper, we regard the MCS, general registers and main memory as *Core State* due to their importance in live forensic acquisition. According to this definition, previous live forensic approaches apparently have more or less shortages in acquiring core state. For example, previous virtualization based live forensic method changes the MCS significantly during installation, and prior in-OS live forensic methods cannot assure the accuracy of the cloned core state. Even worse, both of them focus on main memory only and need suspending the target system to obtain an accurate result. It is worth noting that Vis only focuses on accurately retrieving target system's core state currently. The main reason is that supporting disk cloning in live forensic tools has no impact on the original design, only requiring additional engineering effort. And also, much previous work on static forensic [9, 10, 13] focuses on how to dump disk content. Hence, a system designed for how to obtain target system's core state accurately can also ensure the accuracy of dumping the target system's machine state.

### 2.2 Goals and Assumptions

Two main design goals are identified to provide accurate core state acquisition without suspending the target system. First, the proposed techniques should cause no modification to MCS. By constructing execution environment for live forensic tools with no influence to MCS, it is possible to obtain an accurate MCS copy during acquisition. And more important, no host environment impact makes it come true to continue running the target system when loading and unloading Vis acquisition tool. Second, the proposed techniques should not write to the result image with any content which is produced after the acquisition time point. The challenge of this goal is that the proposed techniques should be able to identify which part of core state content is newly generated and point out what is the original content in that location. In this way, the proposed live forensic tool owns the ability to dig out the evidence which is possibly contaminated, or even lost from the result produced by previous forensic tools.

***Assumption*** Loading as an OS driver, Vis shares the same usage model with many existing forensic tools listed in [35]. Occupying certain amount of memory is necessary according to the Locard's exchange princi-

ple [15]. We assume this does not impact on the existence of critical evidence. It is acceptable since this usage model is popular in a large amount of modern live forensic tools [35]. Moreover, Vis leverages the isolation perspective in virtualization when loaded and hence has no impact on machine state. In the mean time, we assume Vis is started from commercial OS on non-virtualized environment. We do not consider the recursive virtualization situation due to the lack of hardware virtualization support inside guest virtual machine. Vis theoretically works on fully nested virtualization environment.

## 2.3 Vis Design

Based on Vis's design goal, we propose two key techniques of Late-Virtualization and Virtual-Snapshot, to fulfill the design requirement. Next, we will describe how the proposed technologies achieve Vis's corresponding design goals.

### 2.3.1 Late-Virtualization Approach

In any live acquisition approach, the tool should provide itself suitable environment to produce trustworthy result. As described earlier, forensic examiners need to obtain target system's core state to form the idea of what has happened. Since acquisition environment decides a tool introspection capability and the impact to the target system during obtaining, a suitable environment helps the tool produce a reliable result. Hence, an ideal environment should fulfill the following two requirements: Space Isolation and Introspection. Space Isolation means the acquisition tools should not violate the target system's machine state integrity. Unfortunately, according to Locard's exchange principle [15], it needs balance between reliable observing and retaining integrity. Hence, practical live acquisition tools, particularly those in-OS ones, always minimize their impact on target system integrity by decreasing usage of the unallocated memory. Introspection means the environment is capable of reliably intercepting certain types of important events, and has a superior or at least the same view of MCS compared with the target system.

Late-Virtualization fulfills the Space Isolation and Introspection requirements by leveraging virtualization technology. It is worth noting that because of the widely support of hardware virtualization on commercial x86 processors currently, our discussion will mainly focus on hardware virtualization which is also needed in Vis's implementation. Hardware virtualization use the "trap and emulate" method [4] to provide abstract hardware interface to guest OS. Available since 2005, commercial x86 hardware solutions enable hypervisor to handle certain OS-related critical events in a first-hand manner. Moreover, by booting hypervisor prior than OS, it allows hypervisor to manage and reallocate available resources. In this way, hypervisor, isolated from guest OS and applications, is still capable of monitoring the whole guest machine.

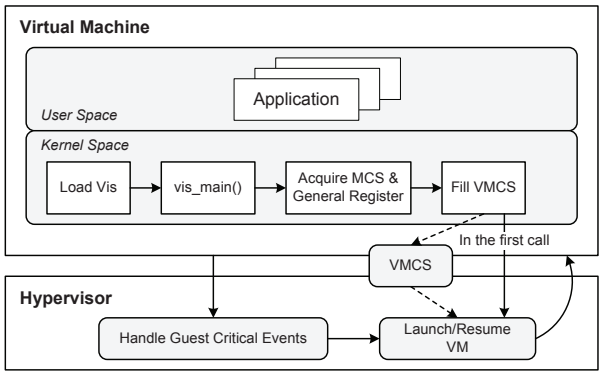


Figure 3: **Late-Virtualization Procedure.** This figure shows the key steps in Late-Virtualization to wrap the target system into a virtual machine. Also, it shows how to handle traps and then resume running the target system.

Hardware virtualization's characteristic decides its suitableness in live acquisition. However, there is one notable pitfall. In traditional ways, rebooting is needed to enable virtualization. It is intolerable in live acquisition practices since rebooting significantly jeopardizes the content in volatile memory and registers. Though prior work [34] claims to have solved this problem, it is still reported that it sacrifices the target system's machine state integrity at a cost. In short, previous virtualization approaches fail to fulfill the Space Isolation requirements. Inspired by NewBluePill Project [24], Late-Virtualization is proposed to completely solve this issue.

Figure 3 demonstrates Late-Virtualization's key procedure in creating hypervisor and used by hypervisor in later running. The two steps, Acquire MCS and General Registers as well as Fill VMCS, are used to achieve the Space Isolation goal without rebooting. Acquiring MCS and General Registers is used to supply necessary architecture information in constructing VM with the same MCS and general registers later. Besides, part of these architecture information are required to be shadowed to conceal the content on real control registers from the view of guest VM. For example, launching hardware virtual machine is required enabling virtual machine mode first. Specifically, CR4 register is modified for this reason during Vis loading. By shadowing CR4 register access from guest VM, the target system can never get the real CR4 register value, and hence no MCS modification is actively incurred by Vis. In order to achieve the goal, firstly it needs to know what the guest VM should

execute at instruction level after creating/resuming execution from hypervisor; secondly it is required to operate atomically to avoid gathering dubious content. To be specific, EIP and ESP registers are needed to decide the first instruction to execute after creating VM. Meanwhile, the length of the current instruction is required to adjust guest EIP when resuming to guest. Also, CR registers contents are also recorded to provide shared page table between hypervisor and guest machine. With shared page table, it is possible to reuse the guest OS drivers to facilitate live acquisition in loose restrictions. What is more, without the need to acquire OS specific information in constructing VM, Late-Virtualization makes Vis independent to legacy OS and applications.

VMCS, referred as *Virtual Machine Control Structure*, is used to define VM execution environment as well as the trapping conditions in hardware virtualization technology. When it turns to the next step to Fill VMCS, Late-Virtualization needs to fill all the recorded system MCS and general registers contents in the corresponding field of VMCS. What is more, Late-Virtualization also needs to register the interested guest critical events as well as their handlers in VMCS. As a result, once these events happen, hardware will automatically store guest state and transfer control to Vis hypervisor before executing any single instruction in guest machine, making it possible for Vis to access all core state of the guest machine. In this way, Late-Virtualization helps Vis hypervisor start up and work while preserving the execution of target system. Since Vis's unloading operation is similar to its loading procedure, Late-Virtualization makes Vis cause no active machine state modification during Vis's lifetime.

### 2.3.2 Virtual-Snapshot Approach

Virtual-Snapshot is used to accurately capture the target system's core state while the target system keeps running. According to the difficulty level of obtaining various scope, core state can be divided into 2 groups. The first group includes MCS and General Registers. The second group includes Main Memory only. Based on Late-Virtualization technology, Virtual-Snapshot is able to obtain an accurate dump of the first group's content by nature. The reason is that during each trap to hypervisor, the control flow will be interrupted by hardware to automatically record guest machine's MCS in VMCS. In addition, Late-Virtualization approach will store the general registers content in its own data space for resuming guest machine execution later. In this way, Virtual-Snapshot only needs to save these information in the indicated file when the corresponding hypercall is issued from Vis client console to achieve the accurate acquisition on guest machine's MCS and General Registers.

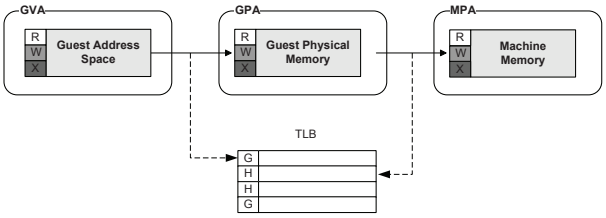


Figure 4: **Nested Paging Mechanism.** This figure describes how GVA is translated into MPA. **G** means that the corresponding TLB entry stores GVA to MPA translation, while **H** means that the TLB entry is used for translating GPA into MPA. **RWX** stands for read, write and execute permissions on specific memory region.

Unfortunately, it is not the same situation for Main Memory. First, it is difficult to monitor modification to all available Main Memory, especially for those in-OS acquisition approaches. Previous virtualization acquisition approaches bypass this problem by suspending target and then obtaining the memory content. Second, the large size of Main Memory causes that a long time is always needed to acquire all memory content. Suppose the target system owns 2GB physical memory, it takes more than 20 seconds to obtain a complete memory dump and output it to the local disk at 100MB/s transfer speed. To make things worse, previous live acquisition approaches need to suspend the machine in order to ensure the result's accuracy. Hence, the required long suspending time makes them inappropriate in stealthy live acquisition occasion.

The first problem is solved by Nested Paging mechanism in Virtual-Snapshot. Figure 4 shows how Nested Paging mechanism translates arbitrary Guest Virtual Address (GVA) into corresponding Machine Physical Address (MPA). In traditional virtualization, Nested Paging mechanism is employed to host multiple VMs on the same physical machine. As a result, a two-level translation mechanism is needed to ensure the compatibility with legacy OSes. Since modern hardware virtualization tries to eliminate the guest OS sense of the underlying hypervisor, the first level address translation, which turns GVA to Guest Physical Address (GPA), still employs the origin guest OS page table pointed by CR3 register as the traditional way does. At the same time, the second level address translation, which uses Nested Page Table (NPT) to translate GPA into MPA, is pointed by Nested Page Table Pointer. It is worth noting that Shadow Page Table (SPT), the traditional software Nested Paging approach which uses another sets of page tables to achieve GVA to MPA translation, can also be employed by Virtual-Snapshot in the situation that hardware assisted nested paging is unsupported on legacy hardware.

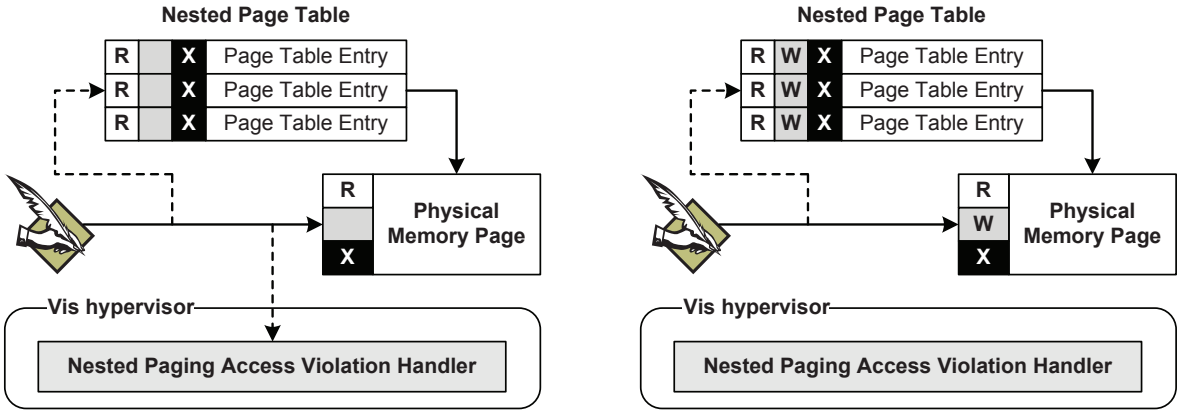


Figure 5: **Virtual-Snapshot Approach.** Comparing with modification to normal pages, a different control flow is executed when modifying write locked pages.

In order to monitor modification on the whole range of target system's Main Memory, Virtual-Snapshot first creates an identical mapping from GPA to MPA on the second level address translation. After that, Virtual-Snapshot actively queries guest OS for its valid physical memory range. This is because certain amount of system memory address space is required for existing I/O devices, e.g., graphic card, network card, etc. In addition, on x64 architecture the valid physical address space is far more tremendous than the maximum supported memory capacity currently. Hence, distinguishing physical memory range from I/O memory range and unallocated memory range helps build core state's border.

After owning the knowledge of a clear physical memory scope during Vis startup, Virtual-Snapshot disables write permission on the whole guest physical memory range when acquisition command is issued from Vis client. As shown in Figure 5(a), achieved by manipulating the second level NPT, revoking the write permission on guest OS physical memory page makes any subsequent writing to this page generate nested page fault before changing any single bit within it. Then, hardware automatically traps to Vis hypervisor to handle it accordingly with hardware generated guest fault frame, which includes the address of the modifying page, the allowed permission as well as the desired permission. The pre-registered nested paging access violation handler in Vis hypervisor dumps the content of the trapped page, removes write lock from the trapped page by regranting the write permission, and then resumes guest machine running from the trapped instruction. Since the guest machine resumes from writing to the same guest physical page again and this time no write lock is put on the same page, the write operation is succeed without interrupting the original information flow, as shown in Figure 5(b). In this way, Virtual-Snapshot obtains the origin content of the guest physical page being modified, while keeping the guest OS and application's information flow, even in the case that Vis is orthogonal to the guest machine.

The second problem is solved in an Amortized manner by Virtual-Snapshot. According to our observation, only a small portion of guest physical pages are modified on each processor during a single instruction execution. Hence, it is sufficient for Vis to dump only the changing pages in order to obtain a complete origin content of guest physical memory. For the remaining comparative huge part of guest physical pages, their dumping are deferred until either their modification or the end of acquisition, if their content is never changed. As a result, by lengthening the necessary acquisition time, Vis has no need to suspend the target system. Even acquisition incurred overhead becomes slight for Virtual-Snapshot dumps small portion of critical pages first and large part of remaining pages later in little pieces.

## 3 Implementation

We have implemented a prototype of Vis and applied it to live acquisition of Windows 7 x86 system. Currently, Vis leverages Intel VT technology [2] to provide the necessary hardware virtualization functionality. The Late-Virtualization is implemented by directly using virtual machine extension to build the underlying hypervisor. For Virtual-Snapshot, we employ Intel's Extended Page Table (EPT) technology [2] (We refer EPT to Extended Page Table instead of the corresponding technology in the following), feasible from CPUs produced after 2008 with Nehalem micro architecture, to enable hardware as-

sisted paging. The engineering effort shows that Vis is a lightweight approach. Vis totally needs 5962 Source Lines of Code (SLOC), involving 871 SLOC for Virtual-Snapshot and 5091 SLOC for Late-Virtualization technique.

During Vis's implementation, it is necessary to balance between Vis's effectiveness and its applicability. There are two groups of alternative decisions: Restoring Timer Stamp Counter (TSC) vs. Non-Restoring TSC and Synchronized Write vs. Asynchronized Write.

## 3.1 Restoring TSC vs. Non-Restoring TSC

The alternative choice between Restoring TSC and Non-Restoring TSC comes from the fact that TSC, which increments its value by 1 atomically after every clock tick, keeps updating itself by hardware even in hypervisor execution. In this case, memory write instruction on write locked guest physical page will update TSC thousands of times. Comparing with its execution in the original target system, it incurs a latency of tens of clock ticks. According to our observation, it needs 196505 clock ticks to handle a single nested paging access violation and perform corresponding dumping task. Vis hypervisor utilizes 5% to 7% overall CPU time when performing live acquisition. Subsequently, this side effect can potentially, though never observed, change the target system's control flow, e.g., causing timeout on waitable locks.

The solution is to record TSC value during every trap to Vis hypervisor (denoted as #VMEXIT) and later restore TSC just before resuming to guest machine (denoted as #VMRESUME). Also, since #VMEXIT and #VMRESUME events cost constant clock ticks to accomplish, Vis hypervisor can adjust backwards the TSC value to mask the corresponding overhead. However, there is one notable pitfall. Although the target system has difficulties in sensing Vis hypervisor existence by checking TSC for instruction execution latency, it is possible to detect Vis acquisition action via external timers, i.e., quartz clock. According to our experience, a 8$\sim$12 seconds latency from external time is observed after finishing a complete Vis live acquisition.

## 3.2 Synchronized Write vs. Asynchronized Write

Another alternative choice, Synchronized Write vs. Asynchronized Write, comes from the need to balance the reliability, performance and the required engineering effort. Previous work [11] concerns that OS file system drivers and disk drivers are unreliable in live acquisition for the reason of possible contamination caused by malicious code. However, a considerable amount of live acquisition tools do not provide their own utility drivers, for example, Win32dd and FAUdd. One benefit is that it can reduce the required engineering effort, and another one is that it can decrease the memory usage to minimize the influence brought in by live acquisition tools to the target system. In Vis implementation, there are two methods available to output the dumped target system's core state to disk. The first one is Synchronized Write, which means writing to result file immediately via static-linked drivers during each nested page access violation; the second one is Asynchronized Write, with the meaning of buffering the original content during each trap to Vis hypervisor, then reusing existing OS drivers and delegating the output task to OS worker thread.

Both of them have several advantages and disadvantages. The biggest advantage of Synchronized Write is that it can produce a more reliable target system's core state image. Since no buffer space is needed, Vis primary memory usage is to hold its code and EPT. It results that Vis requires as much as the memory needed by other live acquisition tools, which is acceptable according to our assumption. Synchronized Write has several disadvantages. Since disk operation is involved during hypervisor execution, it downgrades the target system's performance and raises the Vis hypervisor CPU time during acquisition, leading to a much higher possibility of causing timeout on target system's waitable locks. Besides, a significant engineering effort is needed to implement the necessary drivers, though it has no impact on Vis design. Moreover, it is notable that Synchronized Write via OS legacy drivers is sometimes unattainable for Vis due to OS design. According to our experience, some of the trapped guest memory write operation occurs at a higher interrupt request level than that is required for disk operation on Windows 7 and always results in a Blue Screen of Death.

On the contrary, Asynchronized Write frees Vis from the burden to implement its own required drivers and incurs lower performance impact to the target system via OS I/O scheduling. The biggest disadvantage is that the acquisition result is of less reliability; and the required buffer memory is usually large, depending on the characteristics of workload. Since applying Asynchronized Write incurs no design modification as we described before. The current Vis implementation employs Asynchronized Write as its output technique. And Vis allows only local disk can be used to store acquisition result. Meanwhile, 1GB out of 2GB physical memory is reserved for buffering. How to reduce the buffer memory size and ensure the accuracy of obtaining origin buffer content are left for future work.

Moreover, both Synchronized Write and Asynchronized Write need to concern certain implementation issue to achieve Vis functionality. Firstly, some guest physical memory pages are never changed since the tar-

get OS startup, e.g., most of the OS code pages. Relying on the dumping operations in the access violation handler, Vis can not obtain the content in these pages. To solve this problem, Virtual-Snapshot is configured to dump the remaining pages in an amortized manner when other guest machine event handlers in Vis is triggered. In Vis current prototype, Virtual-Snapshot also dumps the remaining pages from low page frame number when the target OS tries to write CR3 register, which is a frequent operation on multitasking OS on x86/x64 architecture. Secondly, whenever nested paging access violation occurs, it is required to set the write permission to the corresponding page before resuming the guest machine. Otherwise it will cause infinite trapping. Hence, the buffer memory needed by the access violation handler is identified as *Critical Buffer Space*, and a *Non Critical Buffer Percentage (NCB%)* is needed to avoid too much buffer memory used by the dumping operations performed in other handlers. NCB% is defined as the most percentage of buffer memory to be used by the dumping operations in other handlers. Generally, more pages dumped within a single trap or less critical buffer space reserved for dumping operation in the access violation handler leads to a possible violation of Vis acquisition accuracy.

### 3.3 Vis Optimization

Vis has gained three optimizations in the current prototype. The first optimization shortens the acquisition time. During the implementation of Asynchronized Write, we noticed that it is unnecessary to limit Vis from dumping one page per trap. The acquisition accuracy is still guaranteed even if multiple pages are obtained at once, which also shortened the acquisition time. The reason is that obtaining multiple pages at once will reduce the total trap times. Hence, less overhead is incurred on the context switches between hypervisor and VM. Also, a larger I/O buffer is needed when acquisition several pages per trap is enabled. The I/O performance improves greatly when I/O buffer increases within a certain range, resulting in decreasing Vis acquisition time in advance. Though shortening Vis acquisition time a lot, this optimization has one disadvantage. Dumping multiple pages per trap results that Vis hypervisor spends more time to handle a single trap. Generally, the more pages dumped during each trap, the shorter the time is required to accomplish live acquisition, and the higher possibility is to incur side effect to the target system. We experiment dumping 8, 16, 32, 64 pages within single trap. In Section 4, we will evaluate the effectiveness and performance under these conditions.

The second optimization decreases Vis startup time. During Vis loading, Virtual-Snapshot needs to build EPT, the four level page table, with identical mapping from GPA to MPA. In the earlier version of Vis, building identical mapping is done with a one-by-one page frame mapping. Hence, it always needs to traverse the same EPT structures and set different entries on the last level of EPT. As a result, Vis requires about 8 seconds to construct the EPT to finish the mapping task. This optimization is to batch every 512 mappings within a single operation. Thus, it significantly reduced the times of walking EPT. In the current Vis prototype, the loading time of current Vis prototype is imperceptible.

The third optimization is more related to engineering. In order to be more flexible, we create the idle state for Vis. Vis is implemented to stay in the idle state and transit to acquisition state if and only if the acquisition command is issued from Vis client. After the acquisition is accomplished, Vis transits to the idle state again. In idle state, Vis does not intercept any write attempt to guest physical memory though EPT is still enabled. Hence, no single bit in guest physical memory is dumped and outputted to local disk and the performance impact to the target machine is minimized.

## 4 Evaluation

The current Vis implementation realizes its design goals described in earlier sections on Windows 7. All experiments are conducted on a Dell Optiplex 980MT host with a 3.2GHz Intel i5-650 processor, 2GB RAM and a gigabit ethernet card. We use the uniprocessor x86 version of Windows 7 in our experiments. In this section, we first analytically examine the accurate live acquisition guarantees provided by Vis. Then we present its overall performance as well as the performance impact on the target system.

### 4.1 Effectiveness Evaluation

In Section 1, we compare the acquisition accuracy evaluation of Vis with another two commonly studied live forensic tools, as shown in Figure 1. The experiment methodology is that we load the acquisition process first, and then manually start pollution process immediately after beginning acquisition. The pollution process allocates and fills memory with unique content that will be nonexistent if not polluting. As Figure 1 shows that, even in the situation that the pollution process allocates and pollutes memory at the rate of 2500 pages per second for 20 seconds, no single polluted page is dumped by Vis with the target running. On the contrary, though Win32dd tool finishes its dumping physical memory task in 17 seconds and Memoryze, 18 seconds, they recorded 71.62% and 56.96% polluted pages in the result file respectively. In addition, both Win32dd and Memoryze acquisition tool are not able to obtain MCS, or at least

| | Pages Dumped Per Trap | | | |
|---|---|---|---|---|
| | 8 | 16 | 32 | 64 |
| NCB% = 0% | | | | |
| Time (s) | N/A | N/A | N/A | N/A |
| Missing pages | N/A | N/A | N/A | N/A |
| Disk write (KB/s) | N/A | N/A | N/A | N/A |
| NCB% = 20% | | | | |
| Time (s) | 105.86 | 77.90 | 66.92 | 55.06 |
| Missing pages | 0 | 5650 | 29402 | 63089 |
| Disk write (KB/s) | 19101.9 | 25959.6 | 30217.3 | 36728.6 |
| NCB% = 40% | | | | |
| Time (s) | 101.05 | 76.08 | 62.60 | 51.87 |
| Missing pages | 0 | 10228 | 35327 | 61343 |
| Disk write (KB/s) | 20011.9 | 26579.3 | 32305.6 | 38986.0 |
| NCB% = 60% | | | | |
| Time (s) | 97.09 | 72.57 | 58.76 | 49.07 |
| Missing pages | 0 | 14185 | 33242 | 59526 |
| Disk write (KB/s) | 20829.0 | 27864.0 | 34412.3 | 41210.6 |
| NCB% = 80% | | | | |
| Time (s) | 90.20 | 69.00 | 53.63 | 44.75 |
| Missing pages | 599 | 18377 | 39028 | 65202 |
| Disk write (KB/s) | 22420.1 | 29307.3 | 37705.2 | 45193.0 |
| NCB% = 100% | | | | |
| Time (s) | 85.47 | 63.86 | 51.57 | 43.01 |
| Missing pages | 5364 | 21974 | 38133 | 72508 |
| Disk write (KB/s) | 23658.7 | 31665.2 | 39214.3 | 48136.3 |

Table 1: **Vis's Overall Performance.** This figure compares Vis overall performance under different configurations when enabling Non-Restoring TSC. It is noteworthy that the result is not available when NCB% = 0%.

there are no claims about it. In comparison, Vis can retrieve MCS and assure its accuracy in theory.

To the best of our knowledge, Vis is the first system that is able to provide accurate live acquisition while the target native system keeps running. This guarantee is achieved by Late-Virtualization's isolation and introspection characteristics as well as Virtual-Snapshot's amortized obtaining manner. With hardware virtualization support, Vis always obtains the target system's original content before any subsequent modification.

## 4.2 Overall Performance

In order to evaluate Vis overall performance in acquisition state, a series of experiments are conducted under different configurations, including the number of pages dumped in each trap, NCB% and Restoring/Non-Restoring TSC.

Vis shows different overall performance under various configurations, as shown in Table 1. In these experiments, Vis's performance under each configuration is tested for 5 times, then the average value is recorded as the final result. After analyzing Table 1 thoroughly, we can see that it firstly shows Vis can accurately obtain the original memory content in the target system by dumping 8 pages in each trap and being configured with NCB% = 60%. Secondly, with the same number of pages dumped in each trap, Vis acquisition time decreases when the NCB% raises. However, the result is not available when Vis is configured with NCB% = 0%. With this configuration, Vis can not obtain the content in those never changed pages because no buffer space can be used outside Vis's access violation handler. Hence, Vis requires 97.09∼105.86 seconds for accurate acquisition, in the case of dumping 8 pages per trap with NCB% varies from 20% to 60%. Thirdly, with the same NCB%, Vis decreases its acquisition time by increasing the number of pages dumped per trap, which proves the claim that the more pages obtained per trap, the less time is needed in Vis acquisition. Also, the trend of Missing Pages proves that dumping more pages per trap or configuring a higher NCB% leads to Vis acquisition accuracy violation, as described in Section 3.

| | Pages Dumped Per Trap | | | |
|---|---|---|---|---|
| | 8 | 16 | 32 | 64 |
| Average | 338.62% | 143.83% | 52.58% | 18.15% |

Table 2: **The Ratio of Range to Average of Missing Pages.** We calculate the ratio of range to average of missing pages under different configurations. Then, we record the average value of every data column as the final result.

From the experiment results, we also observed two noteworthy symptoms. The first symptom is that Vis has a smaller Missing Pages number with NCB% = 60% than that with NCB% = 40% under the configuration of dumping 32 pages per trap, as shown in Table 1. Also, Table 2 shows the recorded Missing Pages results have a wide range under the same configuration. It is normal because non-deterministic events such as interruptions lead to various amount of modification to the target system's code and data portions. All the modifying pages are required to be dumped before overwriting their original content. As a result, a large amount of modification pages exhaust critical buffer space more easily, leading to acquisition accuracy violation. Another reason is that NCB% is defined as the most percentage of buffer space which can be used except the access violation handler. It is possible for the access violation handler to take any proportion of the buffer space, since the access violation handler always has a higher priority in using Vis's buffer

| Speed Mode | Buf. Size | Time | CPU Util. | Disk Write |
|---|---|---|---|---|
| Normal | 4 KB | 76 s | 46.18% | 26607.95 KB/s |
| Fast | 64 KB | 18 s | 19.59% | 112344.67 KB/s |
| Sonic | 512 KB | 17 s | 6.00% | 118953.18 KB/s |
| Hyper Sonic | 1024 KB | 17 s | 5.36% | 118953.18 KB/s |

Table 3: **Win32dd Performance.** When smaller than 512KB, the I/O disk buffer is the major effect to the disk write thoughput.

space.

The second symptom is that the ranges of Missing Pages become smaller in average when Vis dumps more pages per trap, as shown in Table 2. The reason comes from both the memory layout and its space locality on Windows 7. Because kernel code and data always start from the low physical address space in Windows 7 x86 version. The kernel memory is more likely to be dumped in Vis's guest CR3 write handler. As a result, the impact caused by non-deterministic events is minimized and the Missing Pages result is more stable among different runs.

We also compare Vis and Win32dd acquisition performance. Table 3 shows the Win32dd acquisition performance under 4 speed modes, which only leads to different I/O buffer size according to the technology support of Win32dd. The elapsed time we recorded is directly obtained from Win32dd acquisition report, while the average CPU utilization is retrieved by means of Windows Management Instrumentation (WMI) [39]. The Win32dd acquisition performance evaluation shows that it needs 17∼76 seconds for a complete live acquisition and the more I/O buffer space the less acquisition time as well as cpu utilization is needed. Also, the result shows that the performance improvement is imperceptible when the I/O buffer space is larger than 512KB due to hardware limitation. With the same experiment methodology, Vis evaluation result shows that it incurs 4.74% to 6.48% cpu utilization in average under different NCB% when dumping 16 pages per trap. Considering the acquisition time with this configuration shown in Table 1, it can be calculated that Vis results in 20.5% to 23.1% cpu utilization without amortizing I/O operations in theory, suggesting Vis main acquisition performance impact is caused by I/O operations when comparing this result with Win32dd cpu utilization in fast mode. In addition, though taking 39.3%∼471.1% more time than Win32dd to accomplish live acquisition, Vis is applicable when comparing its acquisition with that of real world static forensic tools, which needs hours or even days to obtain a complete dump result.

After enabling Restoring TSC configuration in Vis, it shows a decrease in acquisition time in all situations, as shown in Table 4. In average, the acquisition time is decreased by 7.30% compared with that configured Non-Restoring TSC. Besides, Table 4 shows that the more pages dumped per trap, the smaller the difference in acquisition time between Restoring/Non-Restoring TSC. This is obvious since the more pages dumped per trap, the less traps are needed for a complete acquisition to the same target system. Since the total I/O operation takes a constant time for the same target system, less traps decrease the total time spent on context switch between guest machine and Vis hypervisor. It is worth mentioning that this acquisition time difference only reflects querying system time by means of reading local TSC. Restoring TSC has no effect on external timer, as described in Section 3.

| **NCB%** | **Pages Dumped Per Trap** | | | |
|---|---|---|---|---|
| | 8 | 16 | 32 | 64 |
| 0% | N/A | N/A | N/A | N/A |
| 20% | 8.84 s | 5.43 s | 6.44 s | 4.91 s |
| 40% | 7.23 s | 5.86 s | 5.14 s | 3.91 s |
| 60% | 7.27 s | 5.24 s | 5.10 s | 3.95 s |
| 80% | 5.82 s | 4.41 s | 4.08 s | 2.45 s |
| 100% | 5.60 s | 4.29 s | 3.60 s | 3.48 s |

Table 4: **Decreased Acquisition Time (seconds) by Enabling Restoring TSC.**

| | On Acquisition | Idle |
|---|---|---|
| Scenario 1: Read CR3 | | |
| #VMEXIT world switch | 966 | 777 |
| Read CR3 value | 316 | 179 |
| #VMResume world switch | 1355 | 760 |
| Scenario 2: Write CR3 | | |
| #VMEXIT world switch | 966 | 548 |
| Write CR3 value | 113 | 113 |
| Handle dumping | 214934 | N/A |
| #VMResume world switch | 1355 | 760 |
| Scenario 3: Handle EPT Violation | | |
| #VMEXIT world switch | 919 | N/A |
| Clone origin page contents | 36232 | N/A |
| Reset EPT entry | 157112 | N/A |
| #VMResume world switch | 2243 | N/A |

Table 5: **Vis Micro Analysis.** This figure shows the needed clock ticks in handling Read/Write CR3 and EPT violation happens in the target system.

## 4.3 Performance of Legacy Software

***Micro Analysis*** Table 5 presents the micro analysis that measures the overhead of handling Reading/Writing CR3 and handling EPT violations, both including acqui-

sition state and idle state. In these experiments, Vis is configured to dump 8 pages per trap with NCB% = 20%. We perform a complete live acquisition as well as keep Vis in idle state for the same time length. Hence, the Guest Read CR3 scenario has happened for hundreds of times, and the other scenarios have happened for tens of thousands of times. Then, the average is recorded as the final result. In Table 5, #VMEXIT World Switch means the total clock ticks spent on hardware context switch and delegating event to the proper handler; #VMResume World Switch means the total clock ticks needed for both necessary cleaning work and hardware resuming VM. In these experiments, all the benchmarks exhibit low overhead in context switch between the target system and Vis. Besides, there is no EPT violation handling in Vis idle state because write attempt interception is disabled in idle state.

During Vis acquisition, the Handle Dumping value item takes the most clock ticks in Table 5. The reason is that it includes both the Clone Origin Page Contents and Reset EPT Entry functionality, as well as other functions to check whether its own buffer space is exhausted. Reset EPT Entry takes the second most clock ticks, indicating it is possible to improve Vis acquisition performance by adopting better EPT entry resetting algorithm (e.g., 8 entries batching resetting) or waiting for EPT technology to become more mature. Moreover, the more mature EPT technology becomes, the more performance improvement for Vis in idle state gains, especially for the practical scenarios that Vis needs to stay in idle state for a long time before acquisition starts.

***Application Macrobenchmark*** In the former subsection, we have evaluated and analyzed Vis performance in acquisition state. For a complete performance evaluation, we also measure Vis in idle state runtime overhead to target system. We execute cpu-intensive, I/O-intensive benchmarks with Vis. For cpu-intensive applications, we use the SPECint 2006 suite. For I/O-intensive applications, we select IOMeter [1], netperf [2] and Apache web server.

For IOMeter, we perform sequential read (sread), sequential write (swrite), random read (rread) and random write (rwrite) with 512KB buffer. For netperf, we use the Vis running system as the netperf server, and run both TCP_STREAM (net_tcp) and UDP_STREAM (net_udp) benchmarks to evaluate network performance. The Apache web server (httpd) is also deployed on Vis running system, hosting the test website with 8 random files, the size of which varies from 4KB to 16MB. Http_load [3] is a flexible program that parallel performs HTTP requests and can do operations on the requested files to verify their safe arrival. Hence, http_load tool is used in Apache web server benchmark to parallel perform maximum 120 transactions with 120 seconds duration.

[1] http://www.iometer.org/
[2] http://www.netperf.org/
[3] http://www.acme.com/software/http_load/

| Item | Overhead (%) | |
|---|---|---|
| | Vis - No Restoring TSC | Vis - Restoring TSC |
| perlbench | 4.74% | 4.51% |
| bzip2 | 3.03% | 3.20% |
| gcc | 14.16% | 13.57% |
| mcf | 50.38% | 50.38% |
| gobmk | 0.22% | 0.45% |
| hmmer | 0.24% | 0.24% |
| sjeng | 6.13% | 6.32% |
| libquantum | 6.34% | 6.34% |
| h264ref | 1.91% | 1.91% |
| omnetpp | 15.30% | 15.85% |
| astar | 8.39% | 8.60% |
| xalancbmk | 7.46% | 7.84% |

Table 6: **Vis Performance Impact - SPECint Benchmarks.**

| Benchmark | | Overhead | |
|---|---|---|---|
| | | No Restoring TSC | Restoring TSC |
| IOMeter | sread | 0.45% | 0.04% |
| | swrite | 0.71% | 0.78% |
| | rread | 0.10% | 0.13% |
| | rwrite | 0.78% | 0.91% |
| Netperf | net_tcp | -2.09% | -2.10% |
| | net_udp | -0.01% | 0.03% |
| Httpd | throughput | 0.30% | -0.03% |

Table 7: **Vis Performance Impact - I/O Benchmarks.**

The SPECint benchmark result is presented in Table 6. Most of the SPEC benchmarks show less than 6% performance overhead. However, there are three benchmarks with over 10%, and one of them, MCF benchmark, with about 50% overhead. According to our investigation, this overhead is caused by the high EPT TLB Miss frequency during MCF running. On the one hand, arc_t type is 32 bytes long and nr_group variable is always set to be 870 at runtime. When executing the for-statement shown in Figure 6, the **arc** value increments 870 times 32 bytes in each for-loop. This leads to a poor space localization, which then causes higher probability in occupying EPT TLB due to the fact that TLB is shared by both Nested Paging and traditional paging in current implementation of hardware virtualization. On the other hand, every EPT TLB Miss costs the guest machine a maximum 14 times of memory access overhead to complete the nested address translation on x86 platform. This is calculated ac-

```
165  for( ; arc < stop_arcs; arc += nr_group )
166  {
167      if( arc->ident > BASIC )
168      {
169          /* red_cost = bea_compute_red_cost
    (arc); */
170          red_cost = arc->cost - arc->tail->
    potential + arc->head->potential;
            ...
178      }
179  }
```

Figure 6: **A For-Statement in MCF Source Code.** This for-statement is the hottest spot of TLB miss in MCF benchmark. It originally exists in the source code of pbeampp.c

cording to the following facts: traditional page table has two level paging structures, while EPT has four on x86 platform. Hence, the TLB Miss overhead will be greatly amplified when EPT is introduced in.

The I/O benchmark results prove that the overhead brought in by Vis is imperceptible, as shown in Table 7. In theory, Vis with Restoring TSC enabled should have a better performance than that with Non-Restoring TSC configuration. However, the measurement error as well as the low number of times trapping to Vis leads to opposite results recorded in portion of I/O benchmarks. In average, Vis in idle state incurs 9.62% performance impact to the target system when activating Non-Restoring TSC configuration, compared to 9.00% when enabling Restoring TSC. At last, the network throughput result shows it is increased by 1% in average after Vis is loaded, no matter Restoring TSC or Non-Restoring TSC is adopted. We are still investigating on it.

## 5 Related Work

Live acquisition has been studied for several years. Previous live acquisition approaches can be divided into two categories: Software Acquisition and Hardware Acquisition. As introduced in Section 1, in the field of software acquisition, prior approaches include both Virtualization Introspection and In-OS State Fetching. Leveraging Xen to construct isolated introspection environment, VIX tools [23] and Srinivas's work [26] are examples of Virtualization Introspection. Xen developers even propose an on-going project with Copy-on-Write technique to obtain VM state [17]. By running in the Dom0, both of them have no modification to guest system during acquisition in theory. However, all Xen based forensic tools are inapplicable when performing live acquisition on native system, since they bring in a significant environment impact to the target system, including a different hardware interface from that of native devices and modification on IDT, GDT and the Master Boot Record (MBR) on disk. As a result, it needs to reinstall the target native OS and reboot the physical machine after the Xen based live forensic tools are deployed. Vis solves these problems by encapsulating the native system into a single virtual machine after the target OS starts up. Also, shadowing is needed to conceal necessary MCS modification and present their original values to the target system. Ruo's work [6] is another Xen based live forensic tool, which incurs additional environment impact to the target system because it uses guest modules for acquisition.

Another example of Virtualization Introspection is employing process based virtual machine. VMWare Workstation [37] is one representative hypervisor in this type. It provides a means of taking a snapshot of the state of the virtual machine, which includes the whole core state. However, it has the same problem with those Xen based forensics tools. VMWare Workstation needs to suspend the target system during acquisition and provide a different hardware interface to VMs. In short, this technique does not address imaging of the host machine, either.

BodySnatcher [34] uses OS driver to load hypervisor on the fly, which is quite similar to our Late-Virtualization technology. And it uses a built-in acquisition OS to obtain target system volatile memory. The first problem is that in order to dump accurately, BodySnatcher needs to suspend the target system during acquisition, and it is ineffective if the target OS is configured to detect a live analysis attempt as introduced in Section 1. Second, BodySnatcher must expose its modification to target system's MCS, e.g., IDT, in order to keep the target system running and capable of handling critical events in hypervisor. Vis solved the first problem by Virtual-Snapshot, with nested paging and amortized acquisition method, Vis can accurately obtain the target system core state without suspending the target system. Vis has never run into the second problem, because there is no need to modify target system's IDT during Vis's lifetime.

For traditional In-OS live acquisition tools, many of them exist in user level only, e.g., Memoryze [27] and GNU dd [4]. Meanwhile, some of In-OS live acquisition tools are loaded as kernel module. All the acquisition commands are issued from their user level client consoles. Win32dd [30] belongs to this type. The biggest problem for these approaches is that they fail to assure the accuracy of the obtained target system's core state. In Vis, this problem is solved by Virtual-Snapshot. Another feasible accurate live acquisition method is to

[4] http://www.gnu.org/software/software.html

leverage the crash dump facility on modern commercial OSes. This function has been long provided for debugging support. For example, by proper configuration, Windows can generate memory dumps under certain events, e.g., "magic" keystrokes and hardware/software failures. When a failure occurs, OS dumping facility provides necessary information to correct or avoid the error. Kdump [21] provides this functionality by loading a crash dump specific kernel on Linux system. Sun, AIX and HP's UNIX hardware platforms use firmware to achieve crash dumping when coming across special key sequences [1]. However, it is impossible to continue running the target OS or application after a crash dump. Also, the accuracy of the acquisition result is doubtable because rootkits probably hook critical functions in either crash dump modules or filesystem drivers to conceal the important evidences. Finally, alternating the configuration of crash dump facilities requires rebooting machine on some commercial OSes, for example, Windows. As a result, Vis surpasses the approach of employing crash dump facility in live acquisition in both effectiveness and applicableness. Meanwhile, it also shows that Vis can be leveraged in debuggers.

Previous work also proposes hardware acquisition methods to be an alternative method in live acquisition. Currently, these methods rely on accessing target system's main memory through the use of Direct Memory Access (DMA). To achieve this goal, Carrier's work [12] proposes an acquisition specific PCI card which disables CPU and performs DMA operation to obtain target system's volatile memory content. Also, Boileau's work [7] and Martin's work[28] employ firewire protocol to perform DMA operations. This approach seems to have the capability of accurate dumping. However, Rutkowska's work [32] proves that it is probable to present a different memory view to DMA based acquisition devices through configuring Memory Mapped I/O features on emerging chipsets. Hence, the obtained volatile memory content is quite different from the real content present to CPU. Vis does not have this problem because the hardware virtualization technology ensures that hypervisor has a broader view than guest machine. Thus, Vis can access all the target system content within its own context.

In addition, to the best of our knowledge, Vis is the first live acquisition tool capable of obtaining native target system's MCS. Obtaining the target system's core state help forensic examiners form a better understanding of how the main memory content is organized than dumping main memory content only. Also, Vis's ability of accurate acquisition without halting the target system makes it applicable in real world practices.

# 6 Future Work and Conclusion

Though Vis is proved to be practical in real world scenarios, some meaningful extensions are still needed to strengthen its capability. The first extension is attestation of the acquisition result in Vis. Attesting the obtained result ensures its integrity and this is required in live forensic's reporting step. The second extension is permitting result outputted to non-local disk, for example, removable media or remote system. With this feature, it is possible to include accurate dumping disk contents into Vis functionality scope, even allowing live cloning among native systems. The third extension is to protect Vis from being detected, and attacked by malware (like kernel rootkits). As a result, live forensic will benefit from Vis in advance for making the subsequent analysis much easier.

We have presented Vis, a light-weight virtualization approach to provide accurate retrieving of native system state while the target system keeps running. Vis achieves its goal by two key techniques: Late-Virtualization and Virtual-Snapshot. Late-Virtualization is used to provide an isolated running environment for live acquisition tools while having at least the same view with the target system. Also, Late-Virtualization has no impact to target system's MCS by transparently encapsulating the native system into a single virtual machine after the target OS starts up and actively shadowing control registers access from the target system. Virtual-Snapshot is employed to accurately obtain target system's core state at the given time point. It avoids suspending target system during main memory content acquisition by adopting an Amortized acquisition approach. A proof-of-concept prototype has been developed to obtain core state on Windows 7. The evaluation result shows that Vis can reliably provide the intended accurate live acquisition with a small performance overhead.

# References


[1] *PANIC! UNIX System Crash Dump Analysis Handbook*. Prentice Hall PTR, 1995.

[2] *Intel 64 and IA-32 Architectures Software Developer's Manuals*. Intel Corporation, 2007.

[3] ACCESSDATA GROUP. FTK. `http://www.accessdata.com/`.

[4] ADAMS, K., AND AGESEN, O. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 2006), ASPLOS-XII, ACM, pp. 2–13.

[5] ADELSTEIN, F. Live forensics: diagnosing your system without killing it first. *Commun. ACM 49* (February 2006), 63–66.

[6] ANDO, R., KADOBAYASHI, Y., AND SHINODA, Y. Asynchronous pseudo physical memory snapshot and forensics on paravirtualized vmm using split kernel module. In *ICISC* (2007), K.-

H. Nam and G. Rhee, Eds., vol. 4817 of *Lecture Notes in Computer Science*, Springer, pp. 131–143.

[7] BOILEAU, A. Hit by a bus: Physical access attacks with firewire. In *Ruxcon* (2006).

[8] BUCHHOLZ, F. *Pervasive Binding of Labels to System Processes*. PhD thesis, Purdue University, 2005.

[9] BUCHHOLZ, F. P., AND SPAFFORD, E. H. On the role of file system metadata in digital forensics. *Digital Investigation 1*, 4 (2004), 298–309.

[10] CARRIER, B. *File System Forensic Analysis*. Addison-Wesley Professional, 2005.

[11] CARRIER, B. D. Risks of live digital forensic analysis. *Commun. ACM 49*, 2 (2006), 56–61.

[12] CARRIER, B. D., AND GRAND, J. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation 1*, 1 (2004), 50–60.

[13] CASEY, E. *Handbook of computer crime investigation: forensic tools and technology*. Academic Press, 2002.

[14] CHEN, X., GARFINKEL, T., LEWIS, E. C., SUBRAHMANYAM, P., WALDSPURGER, C. A., BONEH, D., DWOSKIN, J., AND PORTS, D. R. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 2008), ASPLOS XIII, ACM, pp. 2–13.

[15] CHISUM, W. J., AND TURVEY, B. E. Evidence dynamics: Locard's exchange principle & crime reconstruction. *Journal of Behavioral Profiling 1* (Janurary 2000).

[16] CITRIX. Xen. `http://www.xen.org/`.

[17] COLP, P., MATTHEWS, C., AIELLO, B., AND WARFIELD, A. Vm snapshots (xen summit 2009). Xen Summit, North America, Feb 2009.

[18] DOLAN-GAVITT, B., SRIVASTAVA, A., TRAYNOR, P., AND GIFFIN, J. T. Robust signatures for kernel data structures. In *ACM Conference on Computer and Communications Security* (2009), E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds., ACM, pp. 566–577.

[19] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 193–206.

[20] GMG SYSTEMS, INC. KnTTools. `http://gmgsystemsinc.com/knttools/`.

[21] GOYAL, V., BIEDERMAN, E. W., AND NELLITHEERTHA, H. Kdump, a kexec based kernel crash dumping mechanism. In *Linux Symposium* (2005).

[22] GUIDANCE SOFTWARE, INC. EnCase. `http://www.guidancesoftware.com/`.

[23] HAY, B., AND NANCE, K. Forensics examination of volatile system data using virtual introspection. *SIGOPS Oper. Syst. Rev. 42* (April 2008), 74–82.

[24] INVISIBLE THINGS LAB. NewBluePill. `http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html`.

[25] JOSHI, A., KING, S. T., DUNLAP, G. W., AND CHEN, P. M. Detecting past and present intrusions through vulnerability-specific predicates. In *Proceedings of the twentieth ACM symposium on Operating systems principles* (New York, NY, USA, 2005), SOSP '05, ACM, pp. 91–104.

[26] KRISHNAN, S., SNOW, K. Z., AND MONROSE, F. Trail of bytes: efficient support for forensic analysis. In *ACM Conference on Computer and Communications Security* (2010), E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds., ACM, pp. 50–60.

[27] MANDIANT CORPORATION. Memoryze. `http://www.mandiant.com/products/free_software/memoryze/`.

[28] MARTIN, A. Firewire memory dump of a windows xp computer: A forensic approach.

[29] MCAFEE, INC. Fport. `http://www.scanwith.com/download/Fport.htm`.

[30] MOONSOLS. Win32dd. `http://moonsols.com/blog/2-blog/9-moonsols-windows-memory-toolkit`.

[31] PEISERT, S., BISHOP, M., AND MARZULLO, K. Computer forensics in forensis. In *Systematic Approaches to Digital Forensic Engineering, 2008. SADFE '08. Third International Workshop on* (May 2008), pp. 102 –122.

[32] RUTKOWSKA, J. Beyond the cpu: Defeating hardware based ram acquisition. In *Blackhat* (2007).

[33] SAVOLDI, A., AND GUBIAN, P. Towards the virtual memory space reconstruction for windows live forensic purposes. In *SADFE* (2008), IEEE Computer Society, pp. 15–22.

[34] SCHATZ, B. Bodysnatcher: Towards reliable volatile memory acquisition by software. *Digital Investigation 4*, Supplement 1 (2007), 126 – 134.

[35] SUTHERLAND, I., EVANS, J., TRYFONAS, T., AND BLYTH, A. Acquiring volatile operating system data tools and techniques. *SIGOPS Oper. Syst. Rev. 42* (April 2008), 65–73.

[36] TA-MIN, R., LITTY, L., AND LIE, D. Splitting interfaces: making trust between applications and operating systems configurable. In *Proceedings of the 7th symposium on Operating systems design and implementation* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 279–292.

[37] VMWARE, INC. VMware Workstation. `http://www.vmware.com/products/workstation/`.

[38] WIKIPEDIA. Digital forensic process. `http://en.wikipedia.org/wiki/Digital_forensic_process`.

[39] WIKIPEDIA. Windows Management Instrumentation. `http://en.wikipedia.org/wiki/Windows_Management_Instrumentation`.