Vis: Virtualization Enhanced Live Forensics Acquisition for Native System

Miao Yu, Zhengwei Qi, Qian Lin, Xianming Zhong, Bingyu Li, Haibing Guan
Shanghai Key Laboratory of Scalable Computing and Systems
Shanghai Jiao Tong University
{ superymk, qizhwei, linqian, zhongxianming, justasmallfish, hbguan } @ sjtu.edu.cn

Abstract

Focusing on obtaining in-memory evidence, current live acquisition efforts either fail to provide accurate native system physical memory acquisition 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 physical memory content while preserving the execution of target native system. Our experimental results indicate that Vis is capable of reliably retrieving an accurate system image. Moreover, Vis accomplishes live acquisition within 97.09∼105.86 seconds, which shows that Vis is much more efficient than previous remote live acquisition tools that take hours and static acquisition that takes days. In average, Vis incurs only 9.62% performance overhead to the target system.

Keywords: Vis, Live acquisition, Accuracy, Virtualization

1. Introduction

After forensic scopes and medias are determined, a typical computer forensics scenario has three steps: acquisition, analyzing and reporting [47, 9]. Focusing on the stages of acquisition and analyzing, computer forensics proposes two key challenges: how to obtain the complete system state and how to analyze the retrieved image effectively [39]. Missing image of memory

content leads to an incomplete or wrong investigation result, even with an incomparable analyzing technology.

Transcending static acquisition strategies, live acquisition extends the information gathering range of forensics examiner, i.e., involving with the volatile data. Considering criminal evidence being stored on permanent I/O device only [10], most static acquisition tools, like Encase [21] and FTK [1], clone disk offline to accurately obtain the evidence. Nevertheless, evidence data existing in volatile memory without disk correspondence are totally beyond the acquisition scope of static acquisition tools. To address such issue, the requirement of live acquisition becomes essential. Live acquisition tools can extract the volatile data in the memory of the target system without blocking it. These data include process information [8], process list [23], kernel objects [16] and raw memory content [4, 41], which may be leveraged to record and reproduce the criminal scene.

Based on the architectural difference, 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 a hypervisor like Xen [5]. VIX tools [23], Ruo's work [4], Srinivas's work [29] and BodySnatcher [42] all belong to this type. The second one is *Non-Virtualization Introspection*. It is designed to obtain indicated volatile system state with a minimal environment impact. Iain et al. [43] list several practical tools for different scenarios, including Win32dd [37], KnTTools [19] and Fport [33]. Memoryze [30] is another popular user process forensic tool of this type.

While owning the ability of unearthing tremendous volume of volatile data, live acquisition also faces significant challenges and risks. The first challenge is that previous virtualization based live acquisition methods alter the system environment significantly [5, 42]. The reason comes from the fact that many previous approaches required loading hypervisor prior to the launching of operating system (OS) [4, 23, 29]. 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, thus causing a great loss of information from volatile memory.

The second challenge raises from the fact that the system is not static [3]. Contents in physical memory change with the running processes, making those previous In-OS live acquisition methods unable to guarantee the accuracy of the retrieved physical memory content at the given time point unless

suspending the machine. However, ideal suspending functionality would require hardware support [22]. Also, it is difficult to dump physical memory accurately by manipulating all page tables via In-OS live acquisition tools, because possible existence of hidden processes makes it tough to actively trace all working page tables. As a result, practical In-OS live acquisition tools, like Win32dd and Memoryze, never consider result accuracy as one of their design goals.

This paper builds on our prior acquisition system, which is named *Vis* [35]. Our previous work presents the design and implementation of Vis, proves its acquisition reliability and evaluates its performance in live acquisition scenarios. In this paper, we try to balance between Vis's effectiveness and its applicability by introducing new idea about Synchronized Write and Asynchronized Write. The experiment result shows that Vis dumps polluted content with inappropriate buffer size. Besides, we propose optimization techniques about minimizing Vis's performance impact to the target OS and depict the corresponding evaluation result. In addition, more related work and more detailed evaluation to Win32dd are included.

The evaluation result shows that even under high pollution rate during the acquisition period, Vis can still ensure the accuracy while preserving the target system execution. Vis is able to retrieve an accurate system image in 105.86 seconds comparing with a range of 17∼76 seconds for Win32dd, 18 minutes for Hypersleuth on a 1Gbps network. Meanwhile, it only incurs 9.62% performance overhead to existing applications. These results prove that Vis owns practical value in real world application.

The rest of the paper is organized as follows. Section 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

We propose two key techniques termed *Late-Virtualization* and *Virtual-Snapshot*, to fulfill the design requirement of Vis.

2.1. Late-Virtualization Approach

Inspired by NewBluePill Project [26], we propose Late-Virtualization technique to insert a light-weight hypervisor after the target OS is started up.

Also, it keeps hypervisor functioning without suspending the target system. Late-Virtualization leverages the wide support of hardware virtualization on commercial x86 processors to fulfill the design goal. In current prototype, Vis employs Intel VT-x technology [25].

Virtualization technique offers hardware emulation and usually provides one or more isolated execution environment on the same physical machine. It is composed of one hypervisor and one or more hosted VMs. It has three essential characteristics: Fidelity, Performance and Safety [2]. With these perspectives, software executes identically in the VM to its execution on native hardware, along with more or less performance overheads. Besides providing the required resource sharing functionality, hypervisor can also be leveraged in security approaches for its interposition capability [44, 28, 13].

Intel VT-x separates the CPU execution into two modes: *VMX root mode* and *VMX non-root mode*. At any time point, CPU runs in only one of the two modes. Software running in hypervisor can supervise guest machine execution by pre-defining the interesting event in Virtual Machine Control Structure(*VMCS*) which contains VM and hypervisor execution environment data. After that, any sensitive instruction executed in a guest VM is interrupted by a VMEXIT event and trapped to the hypervisor. The control flow never returns back unless hypervisor finishes handling the event and then explicitly resumes running the guest machine.

A typical hypervisor launching consists of three steps. First, the VMX root-mode is enabled. Second, the CPU is configured to execute the hypervisor in root-mode. Third, the guests are booted in non-root mode. It worths noting that each core can run at most one hypervisor at any time in current VT-x implementation. Thus we always 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. Since Intel VT-x allows to startup a hypervisor at any time, we change the third step to continue running the virtualized native system in order to delay launching hypervisor.

Figure 1 depicts how the native system environment changes after loading/unloading Vis. During the loading phase, Late-Virtualization builds the required virtualization environment and wraps the target OS into VM on the fly. Loaded as an OS driver, Vis shares the same usage model with many existing forensic tools listed in [43] and requires memory by invoking standard OS APIs. One potential problem is that in order to load the acquisition tool

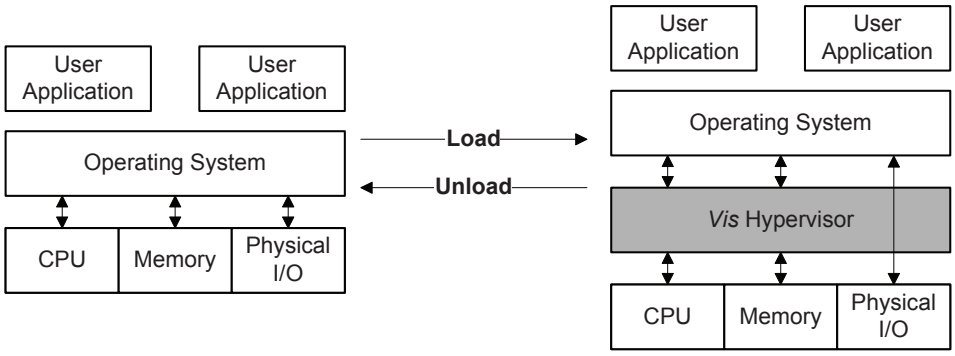

Figure 1: **The Overview of Late-Virtualization Architecture.**

itself, certain memory space is needed and thus there is a potential of jeopardizing evidences in freed memory space. However, most of modern computer architectures require programs to be loaded into physical memory before executing. Vis minimizes its memory usage as other In-OS live acquisition tools do. Also, considering the accurate volatile information gathered from the remaining huge amount of memory, this little memory content modification within Vis installation is acceptable. After all, no zero invasive solution for a *posteriori* forensic analysis exists [31].

One fundamental assumption of Vis is the need for a trustworthy hypervisor. This is shared by many previous research efforts [17, 27, 38]. Even for late-launched hypervisors, previous studies also make the same assumption [31, 42]. Specifically, NewBluePill project [26] discusses how to resist attacks from guest OS by constructing private page table and shadowing control register accesses. Besides, [31] mentions that the hypervisor code can be attested before loading by employing Trusted Platform Module (TPM) [45]. Adopting these security approaches in Vis only requires more engineering effort, lengthens the time needed during loading and incurs slight performance overhead at runtime. In brief, after starting up from commercial OS, Vis assumes the hypervisor is safe enough for live acquisition.

2.2. Virtual-Snapshot Approach

Virtual-Snapshot is used to accurately capture system state without suspending the target system's normal execution. The first group of system state is the register contents. Based on Late-Virtualization technology, Virtual-Snapshot is able to obtain an accurate dump of the register contents by nature. The reason is that during each trap to hypervisor, the control flow

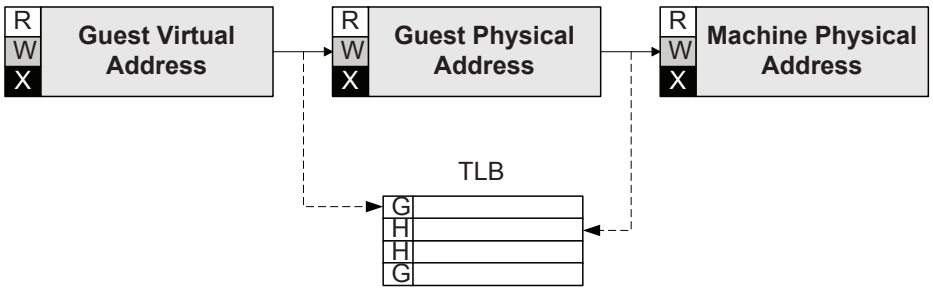

Figure 2: **Nested Paging Mechanism.** This figure describes how GVA is translated into MPA. **G** means that the corresponding TLB entry stores GVA to MPA translation, while **H** means that the TLB entry is used for translating GPA into MPA. **RWX** stands for read, write and execute permissions on specific memory region.

will be interrupted by hardware to automatically record the contents of registers in VMCS. In addition, Late-Virtualization stores the ignored register contents in its own data space as the complement of system state. In this way, Virtual-Snapshot only needs to save these information in the indicated file when performing the live acquisition task.

Unfortunately, it is not the same situation for obtaining the physical memory content. Two challenges are identified in accurately dumping the physical memory content. First, Virtual-Snapshot should be able to identify which part of physical memory content is newly generated and point out what the original content in that location is. Second, the large size of physical memory requires a long time to acquire all the content. Supposing 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. 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 2 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 [18, 34], the first level address

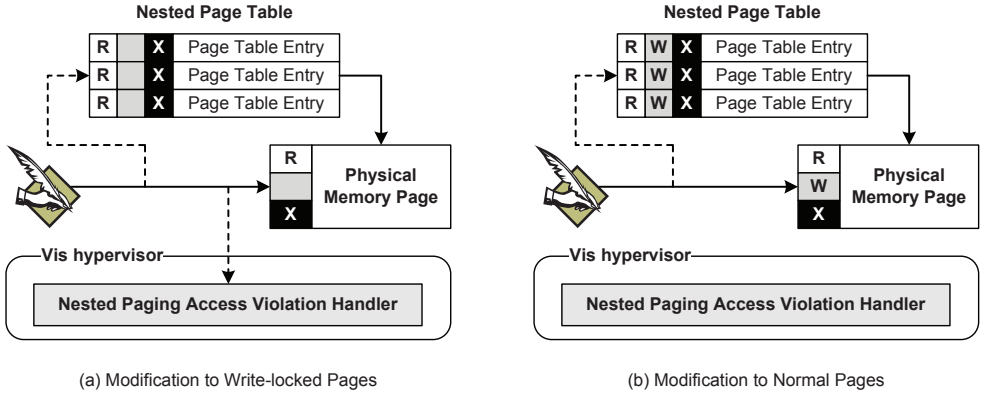

Figure 3: **Virtual-Snapshot Approach.** Comparing with modification to normal pages, a different control flow is executed when modifying write locked pages.

translation, which turns GVA to Guest Physical Address (GPA), still employs the original 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 case that hardware assisted nested paging is unsupported on legacy hardware.

In order to monitor the modification on the whole range of target system's physical 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 by examining kernel data objects. This is because certain amount of system memory address space is reserved 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 the acquisition range in Vis.

After obtaining the knowledge of a clear physical memory scope within Vis startup, Virtual-Snapshot revokes write permission on the whole guest physical memory range when acquisition command is issued from Vis client.

As shown in Figure 3(a), achieved by manipulating the second level NPT, revoking the write permission on guest OS physical memory page forces any subsequent writing to this page to 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 flushes the data cache in order to get persistent view of memory, 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 and this time no write lock is put on the same page, the write operation succeeds without interrupting the original information flow, as shown in Figure 3(b). In this way, Virtual-Snapshot obtains the original 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 by an *amortized* manner of Virtual-Snapshot. According to our investigation, 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 original content of guest physical memory. Dumping the remaining part of guest physical pages is deferred until either their modification or the end of acquisition if their content is never changed. Similar ideas were also suggested by HyperSleuth [31]. Unfortunately, comparing with our approach, HyperSleuth suffered from degenerated performance because it ignores the memory access continuity in its algorithm. Vis improves its acquisition performance by dumping multiple continuous physical memory pages per trap. As a result, by lengthening the necessary acquisition time, Vis does not require to suspend the target system. Furthermore, the acquisition incurred overhead is slight because 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 [25] 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 [25] (We refer EPT to Extended Page Table instead of the corresponding technology in the following), feasible for CPUs produced after 2008 with Nehalem micro architecture, to enable hardware assisted paging. Vis has 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 takes 196,505 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 physical memory content 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 advantages and disadvantages. The biggest advantage of Synchronized Write is that it can produce a more reliable target system's physical memory image. Since no buffer space is needed, Vis primary memory usage is for holding its code and EPT. Therefore, Vis requires a little more memory than other live acquisition tools [43], 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 the 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 operations occur 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 implementing 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 uses only local disk to store acquisition result. Meanwhile, 1GB out of 2GB physical memory is reserved for buffering. How to reduce the buffer memory size and to ensure the accuracy of obtaining origin buffer content is left for future work.

Moreover, both Synchronized Write and Asynchronized Write need to address certain implementation issues to achieve Vis functionality. Firstly, some guest physical memory pages are never changed since the target 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

We propose three optimization techniques in the current prototype. The first one shortens the acquisition time. We notice 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 dumping several pages per trap is enabled. The I/O performance improves greatly when I/O buffer increases within a certain range, resulting in

decreased Vis acquisition time in advance. Though shortening Vis acquisition time a lot, this optimization may cause that Vis hypervisor spends more time on handling a single trap. Generally, the more pages dumped during each trap, the shorter the time is required to accomplish live acquisition, and the higher the 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 batches 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 that we create the idle state for Vis. Vis stays in the idle state and transits 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

We compare the acquisition accuracy evaluation of Vis with another two commonly studied live forensic tools, as shown in Figure 4. The experiment

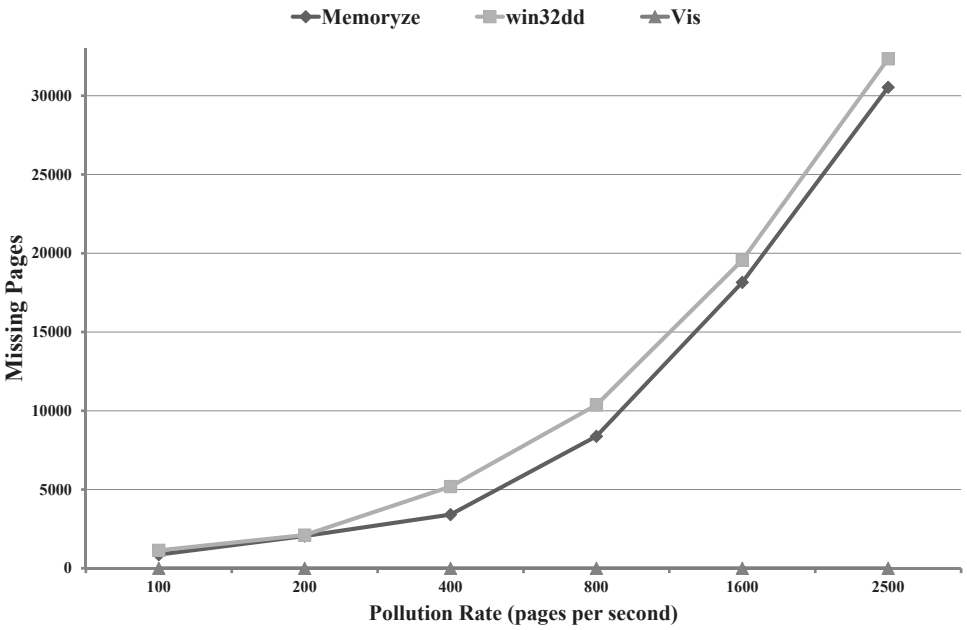

Figure 4: **Accuracy evaluation.** Different page pollution rate is tested for 2GB memory dumping in each test. Missing Pages means the number of the obtained pages containing polluted content.

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 4 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 comparison, Vis can retrieve physical memory image 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.

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

	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	42.01
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%.

	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.

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

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

space, since the access violation handler always has a higher priority in using Vis's buffer 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) [36]. The Win32dd acquisition performance evaluation shows that it takes 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

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.**

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.

	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 Impact

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

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 Translation Look-aside Buffer (TLB) Miss frequency

[1] `http://www.iometer.org/`

[2] `http://www.netperf.org/`

[3] `http://www.acme.com/software/http_load/`

Item	Overhead (%)	
	No Restoring TSC	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.**

during MCF running. On the one hand, arc_t type is 32 bytes long and nr_group variable is always set to 870 at runtime. When executing the for-statement shown in Figure 5, the **arc** value increments 870 times 32 bytes in each for-loop. This leads to a poor space localization, which 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 according 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.

It is noteworthy that similar approaches seem to be able to detect the presence of hypervisor. Hence, they can choose to erase certain memory content afterwards. However, by employing the Blue Chicken anti-detecting strategy proposed in NewBluePill project, Vis hypervisor can notice the unnatural trap frequency by intercepting the reading TSC instruction and return the fake TSC value. As a result, Vis remains hidden from the target system.

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.**

```
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 5: **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

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 and 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 Non-Restoring TSC configuration is activated, compared to 9.00% when enabling Restoring TSC. At last, the network throughput result shows the network throughput 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 [29] are examples of Virtualization Introspection. Xen developers even propose an on-going project with Copy-on-Write technique to obtain VM state [15]. 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 Interrupt Descriptor Table (IDT), Global Descriptor Table (GDT) and the Master Boot Record (MBR) on disk. As a result, they need 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 modification and present their original values to the target system. Ruo's work [4] 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 [46] 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 physical memory content. However, it shares 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 [42] 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 the target system's volatile memory content. The most critical problem of BodySnatcher is that in order to dump accurately, it

needs to suspend the target system during acquisition, thus being ineffective if the target OS has a requirement of 24/7 availability. Another problem is that, BodySnatcher's current implementation exposes its modification 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 physical memory content without suspending the target system. Vis has never run into the second problem, because there is no need to modify the 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 [30] 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 [37] belongs to this type. The biggest problem for these approaches is that they fail to assure the accuracy of the obtained target system's physical memory content. In Vis, this problem is solved by Virtual-Snapshot. Another feasible accurate live acquisition method is to 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 [20] 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 [14]. However, it is impossible to continue running the target OS or application after a crash dump. Also, the accuracy of the acquisition result is dubious because rootkits probably hook critical functions in either crash dump modules or filesystem drivers to conceal 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, the evaluation result also shows that Vis can be leveraged in debuggers.

Previous work also proposes hardware acquisition methods to be an alter-

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

native method in live acquisition. Currently, these methods rely on accessing the 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 the target system's volatile memory content. Also, Boileau's work [7] and Martin's work[32] employ firewire protocol to perform DMA operations. This approach seems to have the capability of accurate dumping. However, Rutkowska's work [40] 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.

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 to be outputted to non-local disk, for example, removable media or remote systems. With this feature, it is possible to include accurate dumping disk contents into Vis functionality, even allowing live cloning among native systems. The third extension is to protect Vis from being detected, and attacked by malware (like kernel rootkits). At last, we will put effort in retrieving the peripheral states of the target system.

Besides, we would also like to extend Vis to support more platforms. Firstly, we'd like to evaluate the chance of adopting Vis on ARM architecture. It is largely employed by smartphones and supports virtualization in recent years [24]. Secondly, Vis can also be used in the cloud environment in case of supporting recursive virtualization [6]. 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 via two key techniques of Late-Virtualization and Virtual-Snapshot. Late-Virtualization is used to provide an isolated run-

ning environment for live acquisition tools after the target OS is started up. Virtual-Snapshot is employed to accurately obtain the target system's physical memory content at the given time point. It avoids suspending the target system during main memory content acquisition by adopting the amortized manner in acquisition. A proof-of-concept prototype has been developed to obtain physical memory content on Windows 7. The evaluation result demonstrates that Vis can reliably provide the intended accurate live acquisition with small performance overhead.

[1] AccessData Group, 2003. FTK. http://www.accessdata.com/.

[2] Adams, K., Agesen, O., 2006. 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. ASPLOS-XII. ACM, New York, NY, USA, pp. 2–13.
URL http://doi.acm.org/10.1145/1168857.1168860

[3] Adelstein, F., February 2006. Live forensics: diagnosing your system without killing it first. Commun. ACM 49, 63–66.

[4] Ando, R., Kadobayashi, Y., Shinoda, Y., 2007. Asynchronous pseudo physical memory snapshot and forensics on paravirtualized vmm using split kernel module. In: Nam, K.-H., Rhee, G. (Eds.), ICISC. Vol. 4817 of Lecture Notes in Computer Science. Springer, pp. 131–143.

[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T. L., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization. In: Scott, M. L., Peterson, L. L. (Eds.), SOSP. ACM, pp. 164–177.

[6] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., Yassour, B.-A., 2010. The turtles project: Design and implementation of nested virtualization. In: Arpaci-Dusseau, R. H., Chen, B. (Eds.), OSDI. USENIX Association, pp. 423–436.

[7] Boileau, A., 2006. Hit by a bus: Physical access attacks with firewire. In: Ruxcon.
URL http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf

[8] Buchholz, F., 2005. Pervasive binding of labels to system processes. PhD thesis, Purdue University.

[9] Carrier, B., 2002. Defining digital forensic examination and analysis tools. International Journal of Digital Evidence 1, 2003.

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

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

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

[13] Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J., Ports, D. R., 2008. 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. ASPLOS XIII. ACM, New York, NY, USA, pp. 2–13.
URL http://doi.acm.org/10.1145/1346281.1346284

[14] Chris Drake, K. B., 1995. PANIC! UNIX System Crash Dump Analysis Handbook. Prentice Hall PTR.

[15] Colp, P., Matthews, C., Aiello, B., Warfield, A., Feb 2009. Vm snapshots. Xen Summit, North America.

[16] Dolan-Gavitt, B., Srivastava, A., Traynor, P., Giffin, J. T., 2009. Robust signatures for kernel data structures. In: Al-Shaer, E., Jha, S., Keromytis, A. D. (Eds.), ACM Conference on Computer and Communications Security. ACM, pp. 566–577.

[17] Garfinkel, T., Rosenblum, M., 2003. A virtual machine introspection based architecture for intrusion detection. In: NDSS. The Internet Society.

[18] Gavrilovska, A., Kumar, S., Raj, H., Schwan, K., Gupta, V., Nathuji, R., Niranjan, R., Ranadive, A., Saraiya, P., Mar 2007. High performance hypervisor architectures: Virtualization in hpc systems. In: Proceedings

of the nineteenth ACM symposium on Operating systems principles. 1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt), in conjunction with EuroSys 2007.

[19] GMG Systems, Inc., 2005. KnTTools. http://gmgsystemsinc.com/knttools/.

[20] Goyal, V., Biederman, E. W., Nellitheertha, H., 2005. Kdump, a kexec based kernel crash dumping mechanism. In: Linux Symposium.

[21] Guidance Software, Inc., 2001. EnCase. http://www.guidancesoftware.com/.

[22] Hay, B., Bishop, M., Nance, K., March 2009. Live analysis: Progress and challenges. Computing in Science and Engg. 7, 30–37.

[23] Hay, B., Nance, K., April 2008. Forensics examination of volatile system data using virtual introspection. SIGOPS Oper. Syst. Rev. 42, 74–82.

[24] Hwang, J.-Y., Suh, S.-B., Heo, S.-K., Park, C.-J., Ryu, J.-M., Park, S.-Y., Kim, C.-R., jan. 2008. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In: Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE. pp. 257 –261.

[25] Intel, I., 2007. Intel 64 and IA-32 Architectures Software Developer's Manuals.

[26] Invisible Things Lab, 2006. NewBluePill. http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html.

[27] Jiang, X., Wang, X., Xu, D., 2007. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In: Ning, P., di Vimercati, S. D. C., Syverson, P. F. (Eds.), ACM Conference on Computer and Communications Security. ACM, pp. 128–138.

[28] Joshi, A., King, S. T., Dunlap, G. W., Chen, P. M., 2005. Detecting past and present intrusions through vulnerability-specific predicates. In: Proceedings of the twentieth ACM symposium on Operating systems principles. SOSP '05. ACM, New York, NY, USA, pp. 91–104.
URL http://doi.acm.org/10.1145/1095810.1095820

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

[30] MANDIANT Corporation, 2008. Memoryze. http://www.mandiant.com/products/free_software/memoryze/.

[31] Martignoni, L., Fattori, A., Paleari, R., Cavallaro, L., 2010. Live and trustworthy forensic analysis of commodity production systems. In: Proceedings of the 13th international conference on Recent advances in intrusion detection. RAID'10. pp. 297–316.

[32] Martin, A., 2007. Firewire memory dump of a windows xp computer: A forensic approach.

[33] McAfee, Inc., 2005. Fport. http://www.scanwith.com/download/Fport.htm.

[34] McCune, J. M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V. D., Perrig, A., 2010. Trustvisor: Efficient tcb reduction and attestation. In: IEEE Symposium on Security and Privacy. IEEE Computer Society, pp. 143–158.

[35] Miao, Y., Qian, L., Bingyu, L., Zhengwei, Q., Haibing, G., 2011. Vis: virtualization enhanced live acquisition for native system. In: Proceedings of the 2nd ACM asia-pacific workshop on systems. APSYS '11.

[36] Microsoft Corp., 1998. Windows Management Instrumentation. http://msdn.microsoft.com/en-us/library/windows/desktop/aa394582%28v=vs.85%29.aspx.

[37] MoonSols, 2008. Win32dd. http://moonsols.com/blog/2-blog/9-moonsols-windows-memory-toolkit.

[38] Payne, B. D., Carbone, M., Sharif, M. I., Lee, W., 2008. Lares: An architecture for secure active monitoring using virtualization. In: IEEE Symposium on Security and Privacy. IEEE Computer Society, pp. 233–247.

[39] Peisert, S., Bishop, M., Marzullo, K., May 2008. Computer forensics in forensis. In: Systematic Approaches to Digital Forensic Engineering, 2008. SADFE '08. Third International Workshop on. pp. 102 –122.

[40] Rutkowska, J., 2007. Beyond the cpu: Defeating hardware based ram acquisition. In: Blackhat.
URL http://www.blackhat.com/presentations/bh-dc-07/Rutkowska/Presentation/bh-dc-07-Rutkowska-up.pdf

[41] Savoldi, A., Gubian, P., 2008. Towards the virtual memory space reconstruction for windows live forensic purposes. In: SADFE. IEEE Computer Society, pp. 15–22.

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

[43] Sutherland, I., Evans, J., Tryfonas, T., Blyth, A., April 2008. Acquiring volatile operating system data tools and techniques. SIGOPS Oper. Syst. Rev. 42, 65–73.

[44] Ta-Min, R., Litty, L., Lie, D., 2006. Splitting interfaces: making trust between applications and operating systems configurable. In: Proceedings of the 7th symposium on Operating systems design and implementation. OSDI '06. USENIX Association, Berkeley, CA, USA, pp. 279–292.
URL http://portal.acm.org/citation.cfm?id=1298455.1298482

[45] Trusted Computing Group, 2011. Trusted Platform Module. http://www.trustedcomputinggroup.org/developers/trusted_platform_module.

[46] VMware, Inc., 1999. VMware Workstation. http://www.vmware.com/products/workstation/.

[47] Yen, P.-H., Yang, C.-H., Ahn, T.-N., 2009. Design and implementation of a live-analysis digital forensic system. In: Proceedings of the 2009 International Conference on Hybrid Information Technology. ICHIT '09. ACM, New York, NY, USA, pp. 239–243.
URL http://doi.acm.org/10.1145/1644993.1645038