Vis: Virtualization Enhanced Live Acquisition for Native System

Miao Yu, Qian Lin, Bingyu Li, Zhengwei Qi, Haibing Guan
Shanghai Key Laboratory of Scalable Computing and Systems
Shanghai Jiao Tong University
{ superymk, linqian, justasmallfish, qizhwei, 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 system. Vis encapsulates the native system into a single virtual machine and then conducts accurate acquisition by manipulating nested page table in hypervisor. We present the design and implementation of Vis, prove its acquisition reliability and evaluate its performance in live acquisition scenarios.

1. INTRODUCTION

A typical computer forensics scenario has three steps: acquisition, analyzing and reporting [18]. 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. The former one is more important because missing evidence 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 the volatile data. Considering criminal evidence being stored on permanent I/O device only [5], most static acquisition tools 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 involve process information [4], process list [9], kernel objects and raw memory content [3], which may be leveraged to record and reproduce the criminal scene.

Based on the architecture 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. VIX tools [9], Ruo's work [3], Srinivas's work [10] and BodySnatcher [16] 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. [17] list several practical tools for different scenarios, including Win32dd, KnTTools and Fport. Memoryze [11] 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. The reason comes from the fact that many previous approaches required loading hypervisor prior to the launching of operating system (OS) [3, 9, 10]. 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 [2]. Contents in physical memory changes 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, providing ideal suspending functionality would require hardware support [8]. As a result, practical In-OS live acquisition tools, like Win32dd and Memoryze, never consider result accuracy as one of their design goals. Also, the acquisition task would take longer when transmitting data over cables. It is reported that BodySnatcher [16] requires suspending the target system for 45 minutes to accomplish a complete 128MB RAM acquisition over 115kbps serial I/O cable. Moreover, it is noteworthy that dumping an accurate physical memory image is difficult by manipulating all page tables for In-OS live acquisition tools, because possible existence of hidden processes makes it tough to actively trace all working page tables.

In this paper, we propose a novel acquisition system named *Vis* to access these challenges. Leveraging virtualization approach, Vis provides accurate retrieving of native system physical memory while preserving the execution of target

system. To validate our approach, we have implemented a proof-of-concept prototype and conducted a series of evaluations. The result shows that even under high pollution rate during the acquisition period, Vis can still ensure the accuracy while preserving the target system execution. Moreover, the performance evaluation result demonstrates that 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 an 1Gbps network. Meanwhile, it 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. Section 3 evaluates Vis through experiments. Finally, we conclude our work in Section 4.

2. DESIGN & IMPLEMENTATION

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

2.1 Late-Virtualization Approach

Late-Virtualization technique is used to insert a lightweight hypervisor after the target OS is started up as well as keep 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 [1].

Intel VT-x separates the CPU execution into two modes: *VMX root mode* and *VMX non-root mode*. At any time point, CPU runs on only one of the two modes. Software running in hypervisor can supervise guest machine execution by pre-defining the interested event in VMCS (*Virtual Machine Control Structure*) 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 traps 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 do not consider the recursive virtualization situation due to the lack of hardware virtualization support inside guest virtual machine. 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 [17] and requires memory by invoking standard OS APIs. One potential problem is that in order to load the acquisition tool itself, certain memory space is needed and thus there is a potential of jeopardizing evidences in freed memory space. Vis meets this issue by claiming that according to the Locard's exchange

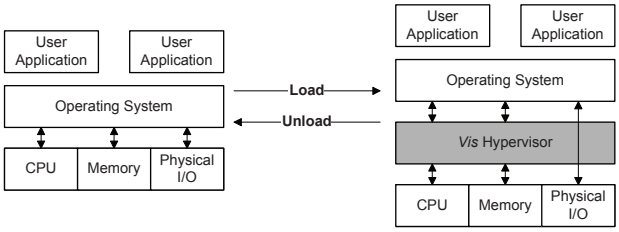

Figure 1: The Overview of Late-Virtualization Architecture.

principle [6], it is inevitable to bring in modification to the observed object. However, considering the accurate volatile information gathered from the remaining memory, this little modification within Vis installation is acceptable. After all, no zero invasive solution for a *posteriori* forensic analysis exists [12].

One fundamental assumption of Vis is the need for a trustworthy hypervisor. This is shared by many previous research efforts [7, 14, 15]. Even for late-launched hypervisors, previous studies also make the same assumption [12, 16]. Specifically, NewBluePill project [12] discusses how to resist attacks from guest OS by constructing private page table and shadowing control register accesses. Besides, [12] mentions that the hypervisor code can be attested before loading by employing Trusted Platform Module (TPM). 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 the content in physical memory without suspending target system's normal execution. 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 causes 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 [13], the first level address 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

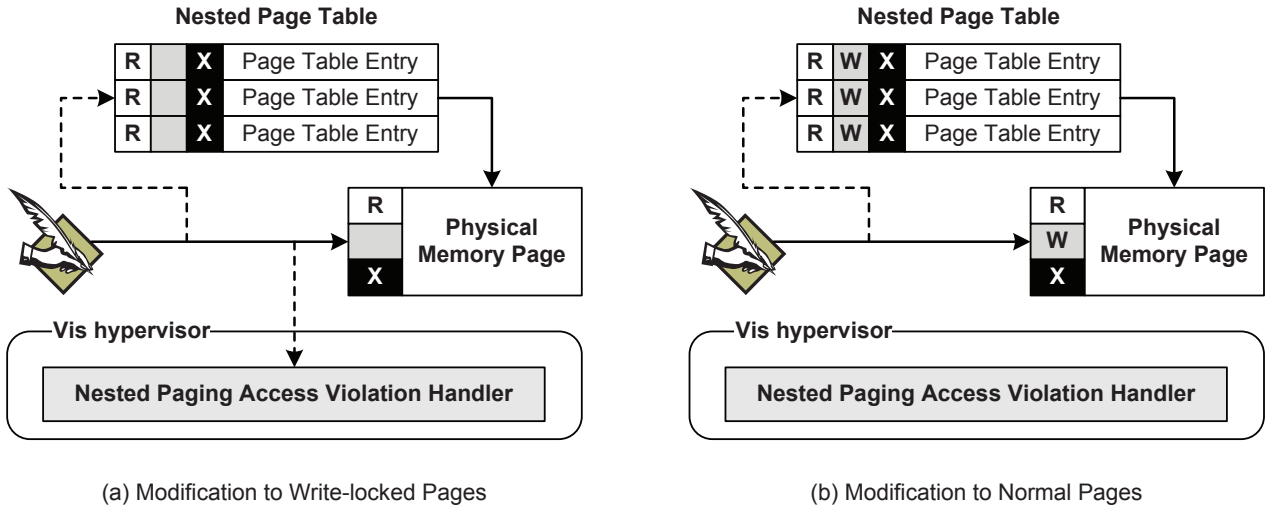

Figure 3: Virtual-Snapshot Approach. Comparing with modification to normal pages, a different control flow is executed when modifying write locked pages.

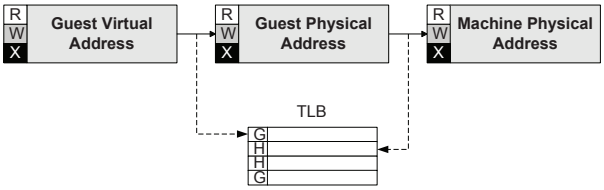

Figure 2: Nested Paging Mechanism. This figure describes how GVA is translated into MPA. G means that the corresponding TLB entry stores GVA to MPA translation, while H means that the TLB entry is used for translating GPA into MPA. RWX stands for read, write and execute permissions on specific memory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

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 again 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 [12]. Unfortunately, comparing with our approach, HyperSleuth sufferred 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 tar-

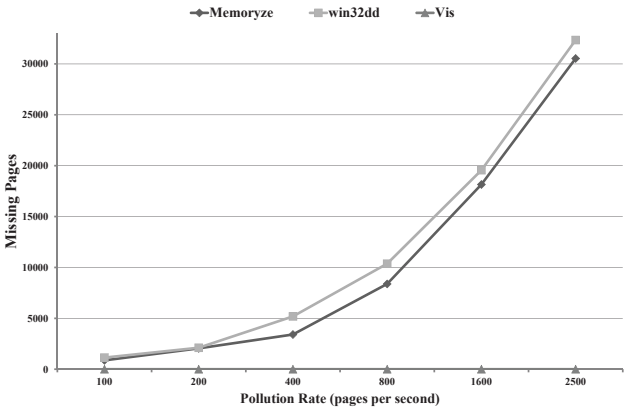

Figure 4: Accuracy evaluation. Different page pollution rate is tested for 2GB memory dumping in each test. Missing Pages means the number of the obtained pages containing polluted content.

get 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. EVALUATION

The current Vis prototype is fully implemented on Windows 7. All experiments were conducted on a host configured 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 begin with verifying the accurate live acquisition guarantees provided by Vis, then present Vis's overall performance as well as the performance impact on the target system.

3.1 Effectiveness

Figure 4 depicts the acquisition accuracy evaluation of Vis with another two commonly studied live forensic tools, Win32dd and Memoryze. All of these tools accomplish live acquisition without suspending the target machine. The experiment methodology is that we load the acquisition process first, and then manually start pollution process immediately after beginning acquisition. The pollution process is used to challenge the accuracy of acquisition result. This is achieved by allocating and filling memory with unique pattern of content. Thus, an ideal live acquisition tool should dump none of the polluted content. As shown in Figure 4, even in the situation that the pollution process allocates and pollutes memory at the rate of 2500 pages per second for 20 seconds, no polluted content is dumped by Vis. On the contrary, though Win32dd finishes its dumping physical memory task in 17 seconds and Memoryze, 18 seconds, they recorded 71.62% and 56.96% polluted content in the result files respectively. The result shows that Vis is able to provide accurate live acquisition.

3.2 Performance Impact

Table 1 presents the micro analysis which measures the overhead of handling writing CR3 and handling EPT violations, both including acquisition state and idle state. In these experiments, we perform a complete live acquisition

	On Acquisition	Idle
Scenario 1: Write CR3		
#VMEXIT world switch	966	548
Write CR3 value	113	113
Handle dumping	214934	N/A
#VMResume world switch	1355	760
Scenario 2: 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 1: Vis Micro Analysis. This figure shows the needed clock ticks in handling Write CR3 as well as EPT violation in the target system.

compared with keeping Vis in idle state for the same time period. Hence, both scenarios have happened for tens of thousands of times during this period. Then, the average is recorded as the final result. In Table 1, "#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. 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 no memory write is intercepted theoretically.

We then compare the performance of Vis with legacy tools in acquisition scenario. Win32dd adopts different I/O buffer sizes to supply four speed modes. The elapsed time we recorded is directly retrieved from Win32dd acquisition report. According to our evaluation, Win32dd requires 17∼76 seconds for a complete live acquisition. In comparison, Vis is able to retrieve an accurate system image in 105.86 seconds when configured to dump 8 pages per trap.

For a complete performance evaluation, we also measure Vis in idle state runtime overhead to target system. CPU intensive and I/O intensive benchmarks were selected to run with Vis so that the quantized result can be evaluated respectively. For CPU intensive applications, we use the SPEC INT 2006 suite. For I/O intensive applications, we select IOMeter, netperf and Apache web server. In average, Vis in idle state incurs 9.62% performance impact to the target system, as presented in Table 2. Besides, the network throughput result shows it is increased by 1% in average after Vis is loaded. Such experimental result is still under investigation.

Although Vis averagely incurs 9.86% performance overhead in SPECint 2006, it is noteworthy that Vis exhibits 50.38% performance impact in MCF benchmark, which implements a graph algorithm. According to our investigation, the total EPT traverse penalty mainly contributes to this overhead. On the one hand, a high EPT Translation Look-aside Buffer (TLB) miss rate is incurred. In MCF's source code, 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 5, the **arc** value increments 870 times 32 bytes in each for-loop. This leads to a poor space locality, which then causes higher probability in occupying EPT TLB due to the fact that TLB is shared by

Benchmark		Overhead
CPU	SPECint 2006	9.86%
I/O	IOMeter	0.51%
	Netperf	-1.05%
	Httpd	0.30%

Table 2: Vis Performance Impact

```
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

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 owns two level paging structures, while EPT owns four on x86 platform. Hence, the TLB Miss overhead would be greatly magnified when EPT is introduced in.

4. CONCLUSION

We proposed Vis, a light-weight virtualization approach to provide accurate retrieving of native system state while the target system stays running. Vis achieves its goal via two key techniques of Late-Virtualization and Virtual-Snapshot. Late-Virtualization is used to provide an isolated running environment for live acquisition tools after the target OS is started up. Virtual-Snapshot is employed to accurately obtain target system's physical memory content at the given time point. It avoids suspending 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.

5. ACKNOWLEDGEMENT

The authors would like to appreciate the anonymous reviewers for their insightful comments on improving this paper's presentation. Also, the authors would like to thank Xiapu Luo and Nguyen Anh Quynh for useful discussions. This work is supported by National Natural Science Foundation of China (Grant No. 60873209, 60970107, 60970108, 61073151), the Key Program for Basic Research of Shanghai (Grant No. 09510701600, 10511500102, 10DZ1500200), IBM SUR Funding and IBM Research-China JP Funding.

6. REFERENCES

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

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

[3] 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.

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

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

[6] Chisum, W. J., and Turvey, B. E. Evidence dynamics: Locard's exchange principle & crime reconstruction. *Journal of Behavioral Profiling 1* (Janurary 2000).

[7] Garfinkel, T., and Rosenblum, M. A virtual machine introspection based architecture for intrusion detection. In *NDSS* (2003), The Internet Society.

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

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

[10] 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.

[11] MANDIANT Corporation. Memoryze. `http://www.mandiant.com/products/free_software/memoryze/`.

[12] Martignoni, L., Fattori, A., Paleari, R., and Cavallaro, L. Live and trustworthy forensic analysis of commodity production systems. In *Proceedings of the 13th international conference on Recent advances in intrusion detection* (Berlin, Heidelberg, 2010), RAID'10, Springer-Verlag, pp. 297–316.

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

[14] Ning, P., di Vimercati, S. D. C., and Syverson, P. F., Eds. *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007* (2007), ACM.

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

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

[17] 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.

[18] Wikipedia. Digital forensic process. `http://en.wikipedia.org/wiki/Digital_forensic_process`.