A Hardware-Based Memory Acquisition Procedure for Digital Investigations *

Brian D. Carrier
carrier@digital-evidence.org

Joe Grand
Grand Idea Studio, Inc.
joe@grandideastudio.com

Abstract

The acquisition of volatile memory from a compromised computer is difficult to perform reliably because the acquisition procedure should not rely on untrusted code, such as the operating system or applications executing on top of it. In this paper, we present a procedure for acquiring volatile memory using a hardware expansion card that can copy memory to an external storage device. The card is installed into a PCI bus slot before an incident occurs and is disabled until a physical switch on the back of the system is pressed. The card cannot easily be detected by an attacker and the acquisition procedure does not rely on untrusted resources. We present general requirements for memory acquisition tools, our acquisition procedure, and the initial results of our hardware implementation of the procedure.

Keywords: Computer Forensics, Digital Evidence, Digital Investigations, Incident Response, Volatile Data Acquisition

1 Introduction

Before digital data can be considered evidence of an incident, it must first be collected. After all, if the investigator does not have a copy of the data, then he cannot extract information from it to draw conclusions. Historically, digital investigations have relied on evidence found on the hard disk, because most of the investigations have involved the storage of contraband data. When investigating computer intrusions, additional data sources are needed and evidence is collected from network traffic and volatile memory.

This paper describes a hardware-based procedure for making an accurate and reliable copy of volatile memory contents so that the data can be examined to find evidence. Two of the more famous examples where the memory contents are useful in an investigation is with the Code Red [1] and SQL Slammer [12] worms. Both of these worms resided only in memory and never wrote anything to disk, so disk analysis may not find any evidence. Other types of investigations could find the memory contents of a suspicious PC interesting because it contains data from running processes, such as passwords, unencrypted data, and the state of user activity [3] [18].

The data stored in volatile memory is lost when power is removed, so it is difficult for an investigator to acquire because he typically wants to acquire data in a trusted environment where there is no threat of malicious programs. Rootkits and Trojan horse attacks against applications and operating system kernels [13] can cause the system to produce unreliable data, so it is desirable to not rely on the resources that the attacker could have modified. Existing methods for acquiring

*Pre-print submitted to the Digital Investigation Journal on December 22, 2003. Published in Issue 1(1), ISSN 1742-2876, February 2004.

volatile memory involve untrusted software and are invasive because they typically write back to memory and may also write to the system's hard disk. Many incident responders will run tools such as `ps` and `netstat` to collect only obvious data, leaving most of the system's memory unanalyzed.

Our solution is a PCI expansion card that is installed into a computer before an intrusion occurs and will dump the exact contents of volatile physical memory to an external, non-volatile storage medium. The PCI controller on the card is disabled until the card is activated by the incident response team and therefore the card will not respond to bus queries from the host system. Only when the device is actually enabled will it become a visible connection to the PCI bus. Due to the design of PCI, which is software- and operating system-independent, the attacker may see the device but may not be able to tamper with its actions.

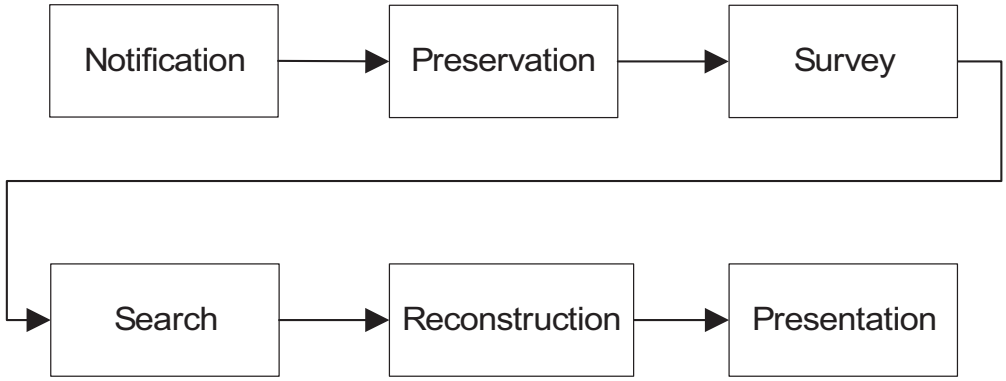
This paper first provides background material on the investigation model that we are assuming and we propose general requirements for memory acquisition. Section 3 shows the existing methods for acquiring the volatile memory contents of a system, Section 4 shows our process for acquiring memory, Section 5 shows the technical details of our implementation, and Section 6 describes the limitations of this process. Finally, we present our conclusions in Section 7.

2 Background

2.1 Investigation Phases

There are several models of a digital investigation, so we will first define the one that we are using [2] and where our proposed procedure fits into it. Figure 1 shows each step of the digital investigation process.

Figure 1: Phases of a digital investigation



The first step of any investigation associated with a computer intrusion is the *Notification Phase*, where an incident is detected and the response and investigation process commences by deploying an incident response team. After the incident has been verified by the incident response team, the investigation moves to a Preservation Phase. The goal of the *Preservation Phase* is to ensure that the contents of the digital crime scene are modified as little as possible so that the investigator can collect accurate evidence. An investigator or first responder will typically make an exact copy of the digital crime scene so that it can be safely analyzed in a laboratory environment.

After the crime scene has been preserved, the *Survey Phase* begins and the crime scene is examined for obvious pieces of digital evidence. This phase uses the existing knowledge of the incident and the experience and training of the investigator to find obvious evidence. After the Survey Phase, the investigator will have one or more hypotheses about the incident and the *Search Phase* will begin with a more thorough search of the system to find additional evidence that

may support or refute his hypotheses. After the digital evidence has been collected during the Survey and Search Phases, the *Reconstruction Phase* begins, in which the existing evidence and hypotheses are tested to form a final theory. In some cases, additional evidence is searched for after the Reconstruction Phase begins. The last phase of the investigation is the *Presentation Phase* where the final theory is presented to the parties who requested the investigation.

Our work focuses on the Preservation Phase, because current techniques do not allow the investigator to preserve the data of all computer components in a reliable way. The investigator can easily make a reliable copy of a hard disk using commonplace tools, but he cannot make an reliable copy of volatile memory. Another goal of our work is to create a mechanism that enables a person with minimal training to preserve a digital crime scene upon detection of an incident until more skilled responders arrive.

2.2 Memory Imaging Requirements

One of the factors that is used when determining the reliability of digital evidence is the acquisition procedure. We have created a set of volatile memory acquisition requirements that can be used when developing a new procedure, when comparing existing procedures, and when developing test cases for implementations of procedures.

The Computer Forensic Tool Testing (CFTT) [14] group at the National Institute of Standards and Technology (NIST) developed a specification for disk imaging tools [15]. The specification lists the requirements that NIST used to test existing disk imaging tools. We modified the disk imaging requirements so that they could be applied to memory imaging. This new set of requirements were used when we designed our hardware-based acquisition solution.

The term "bit-stream duplicate" was removed from the requirements because the term assumes that the source and target data are written in a binary format. While the majority of modern digital systems use a binary storage format, the requirements for acquiring data should apply to any format, although each format may require a different procedure.

1. The acquisition tool shall read all digital data from a source and write them to a non-volatile destination location. The destination, called an image, shall be in an accessible format.

2. The tool shall not cause data to be written to the source.

3. The tool shall follow a documented procedure that includes the steps that it performs and the hardware and software resources that it uses to read the source data.

4. If there are I/O errors while reading the source data, the tool shall write a specified value to the corresponding locations in the image. The tool shall log the type and location of the error in an accessible format.

5. If the destination of the data is larger than the source, the tool shall identify the start and end locations of the source data within the destination.

6. If the destination of the data is smaller than the source, the tool shall notify the user and either abort or copy as much data as possible into the destination.

7. The tool shall provide documentation that is correct.

Software-based solutions for memory imaging typically violate the second requirement because they are loaded into the target system's memory when they execute. The third requirement was added to satisfy a Daubert Hearing [17] and to show what untrusted resources, if any, are used in the

process. If the kernel is used to read the memory contents, then the kernel should be investigated thoroughly to identify if it has been modified in any way to return false values.

The following recommendations are also defined:

1. The tool should halt the target system during the acquisition process so that memory will not change and the page table will remain consistent.

2. The tool should calculate one or more hash values of the data that is read from the source.

3 Previous Work

3.1 Physical Memory Devices

Operating Systems typically provide access to physical memory; for example Unix flavors have the `/dev/mem` device, which corresponds to the physical memory, and the `/dev/kmem` device, which corresponds to the virtual memory of the kernel. Microsoft Windows 2000 and XP have the \\.\`PhysicalMemory` object device that provides access to the physical memory. The `dd` tool, which comes with most Unix flavors and has been ported for Windows systems [7], can read these memory devices and write the data to a file or to a server on the network.

This acquisition procedure is relatively easy to perform, but has some shortcomings with respect to reliability and usefulness. The procedure relies on the local operating system to supply the memory contents and an advanced attacker could modify the operating system to send false data. The procedure also requires the responder to run at least one process. More processes will need to be run if you start a new shell or require network transport. Running a process on the target system violates our second requirement because it will overwrite unallocated data in memory, or allocated data in memory may need to be written to the swap space or page file and thus overwrite unallocated data.

The `/dev/mem` device has been abused by attackers in the past and is restricted in some systems. In fact, some systems do not implement the device at all. Therefore, this technique is not always easily available.

A final shortcoming with this approach, and one that we face with our own procedure, is that the final result is an image of physical memory. Physical memory contains pages of virtual memory that are written in an unorganized order, and the operating system uses a page table to correlate the physical memory and swap space with kernel and process virtual memory locations. Analysis of the page table is required to properly piece together the ordering of contents within physical memory.

3.2 Sparc OpenBoot

The OpenBoot firmware in a Sun system that uses a Sparc architecture can dump the contents of physical memory to a storage device [5]. The firmware can be accessed by using the `L1-A` (or `STOP-A`) keys and the system is suspended and placed into the OpenBoot prompt. Therefore, if an attacker was running any malicious processes, the first responder could suspend the system by enabling the OpenBoot firmware.

If the `sync` command is typed at the OpenBoot prompt, the memory and register contents are dumped to a pre-configured device, typically the swap space on a hard drive. After the memory is written, the system reboots and the `savecore` command can be executed to copy the memory from the dump device to the file system.

The `sync` command was designed to debug the operating system and tries to be efficient with the amount of memory that it saves. By default, it will only save the pages for kernel memory, but the `dumpadm` tool can be used to configure the system to save all memory.

The advantage of this design is that it is hardware-based, due to the fact that the OpenBoot firmware is executing from ROM, and cannot be modified by the attacker (unless the firmware is stored on intentionally rewriteable Flash memory as is common in UltraSparc machines). It also provides a mechanism to suspend a system so that no further activity can occur while a response team travels to a data center. Additional data and symbols from the kernel are also saved with the `sync` command and kernel debugging programs exist to analyze the data.

A disadvantage of this technique is that, by default, it overwrites data in swap space and it requires the system to be rebooted so that `savecore` can copy the memory contents from swap. Another disadvantage of this procedure is that it is available only on Sparc systems. Similar features may exist on other platforms that have firmware-based debugging functionality which can be accessed while the system is running.

Note that other Unix systems also have the `savecore` command, but it can only be run after the kernel panics. In general, there is no graceful way to force a suspect system to panic, so this is not a good procedure to acquire reliable data. Solaris also offers the '-L' flag with `savecore`, which will save the memory of the live system, but this is not much different than using the `/dev/mem` device.

3.3 Process Pseudo-File System

Many Unix systems have a process pseudo-file system that is mounted at `/proc/` which contains information about the kernel and running processes. Each operating system has a different implementation and different directory structure, but many of them have the same basic features. There is typically a file that corresponds to all of physical memory (similar to `/dev/mem`) and a file that corresponds to each of the processes' memory.

We have already discussed the advantages and disadvantages of obtaining the physical memory image with `/dev/mem`, so we will focus on obtaining the memory of a single process. An advantage of this approach is that you will not need to piece the pages of physical memory and swap space together during analysis using the page table. A disadvantage of this approach is that you will need to first identify the suspect processes before this can be used. Another disadvantage of this approach is that it will force non-resident pages of memory to be read from swap and written to physical memory. This may cause resident memory pages to be written to the swap space, which will overwrite unallocated data.

This technique collects only the allocated memory, which is analogous to analyzing a file backup of a compromised server's hard disk as opposed to a full image of it. Passwords and other data from previously run processes may exist in unallocated memory. By running a command to copy the memory, you will overwrite unallocated data in memory because the process will need memory to operate, which violates our second requirement.

The `pcat` tool in The Coroner's Toolkit (TCT) [6] uses the `ptrace()` system call or the `/proc/` file system, if it exists, to save process memory. `pcat` can either save all of the memory as a sparse file or it can save only the non-zero memory contents.

3.4 Virtual Machines

A virtual machine, such as VMWare [19], is an application that emulates a computer environment so that an operating system and other applications can execute inside of it. The operating system

and applications do not always know that they are inside an emulated environment and will execute as normal. The virtual machine can be suspended and saved at any time.

If a system running in a virtual machine is compromised, the memory and disk contents can be saved by suspending the virtual machine and making copies of the files that correspond to the memory and disk areas of the virtual system. Some existing virtual machines save the disk and memory contents in a raw file and others save them in a proprietary format.

The major disadvantage of this approach is the impact on system performance. Critical servers running inside of an emulated environment will increase the load on the hardware. The advantage is that no special processes are run and that trusted software (the virtual machine) is used to save the system state.

3.5 Hibernation

Most portable, and some non-portable, systems have power management features that save the state of the computer while the processor and devices (hard drive, monitor, etc.) are disabled to conserve power. Some implementations provide a small amount of power to the volatile memory in order to retain its contents, while others save the necessary contents of memory to disk and restore the memory when the system is needed again.

Many of the current power management systems use a combination of hardware and software to support the different sleep modes [8]. The operating system can communicate with the BIOS to place the system in a sleep mode or the BIOS can initiate a sleep mode on its own. Unfortunately, many servers are designed to never turn off and therefore may not have a power management option.

If the system state and memory are copied to disk using the power management features of the computer, then this method may be more reliable than the software solutions that we previously discussed. This is similar to the OpenBoot procedure, except the data is written to a dedicated partition and therefore will not overwrite swap space. However, it is not clear if the exact contents of memory that are saved during a hibernation and is unlikely that the entire physical memory is preserved.

3.6 Limitations of Software Procedures

The volatile memory acquisition procedures that are readily available are software-based and rely on untrusted resources, namely the operating system kernel. The user space applications that are used in the acquisition can be run from a trusted device (such as a CD-ROM), but the kernel is always needed to extract the data from memory. There is no way to avoid using the kernel with a software solution because it controls the scheduling of access to the processor and controls all data flow to the storage locations.

A possible solution to the threat of a malicious kernel is to "patch" the needed areas of the kernel memory with trusted code. Kernel patching is a common method used by attackers to install malicious code into kernels. The problem with this solution is that a compromised kernel may intentionally prevent itself from being updated, and the responder will not be able to determine if the kernel is in a state that can be trusted. Simply stated, using an untrusted kernel with software acquisition tools decreases the reliability of the evidence.

A second problem with using a software solution is that it will always require process and kernel memory in order to execute and will therefore overwrite possible evidence. This is analogous to installing disk imaging software directly onto a suspect hard disk before it is acquired.

4 A Hardware-Based Imaging Procedure

4.1 Overview

The goal of our work has been to design and implement a procedure that can make an accurate copy of volatile data and that minimizes the amount of volatile and non-volatile data that is modified on the target system in the process. As discussed in the previous section, we cannot rely on the operating system and software applications to provide reliable data. Therefore, we chose to use a hardware-based solution because it is difficult for the attacker to tamper with, it can access memory without relying on the operating system, and it will not need to use system memory while it is running.

Our procedure uses a PCI expansion card that can be installed into a computer before an incident occurs. When the machine powers up, the acquisition card configures and tests itself, then disables its PCI controller so that it does not respond to bus queries from the host system. Only when the device is actually enabled will it become a visible connection to the PCI bus. Due to the design of PCI, which is software- and operating system-independent, the attacker may see the device but may not be able to tamper with its actions.

The back of the PCI card has a physical switch and an interface to an external storage medium. When the switch is enabled, the PCI controller on the card is activated and takes control of the PCI bus. The card first suspends the CPU (a recommendation which will prevent an attacker or legitimately executing application from modifying memory contents while the acquisition is in process) and then uses Direct Memory Access (DMA) to copy the contents of physical memory to an external non-volatile storage device, such as an IEEE1394 (Firewire) hard disk, Hitachi Microdrive [9], or memory card. Once the physical memory has been successfully copied to the non-volatile storage device, the CPU is resumed and the operating system continues to execute.

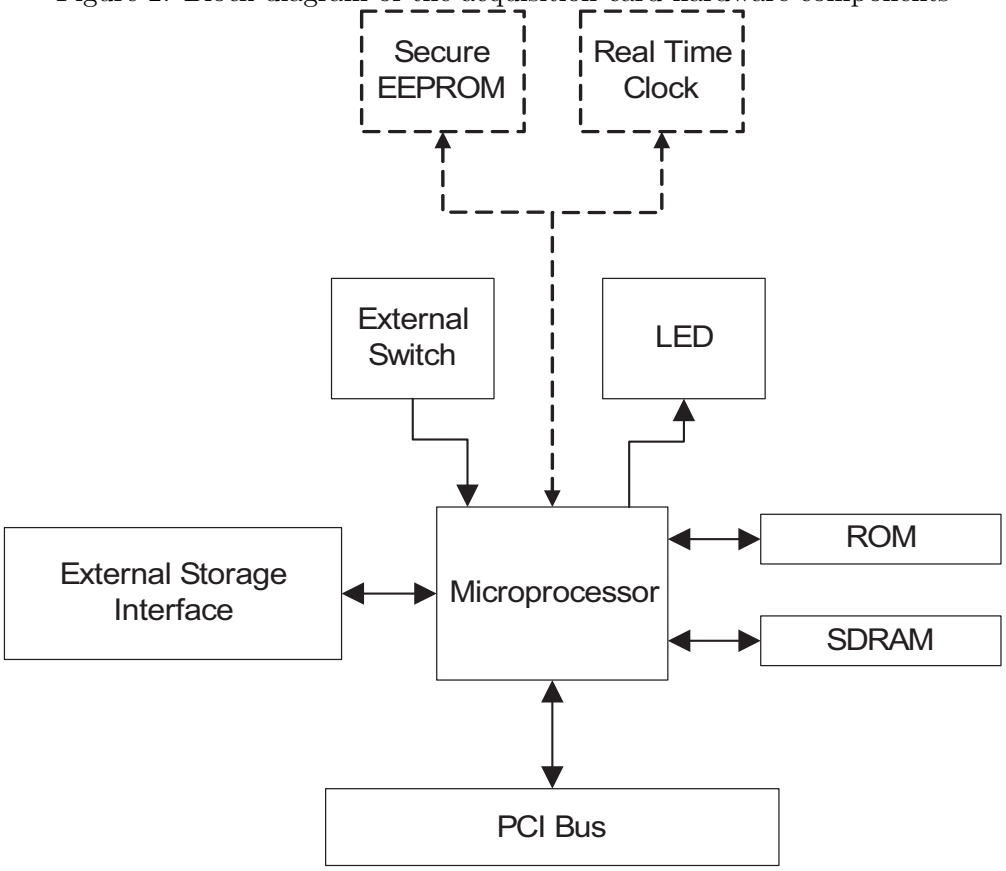
4.2 System-Level Design

This section provides a system-level description of the hardware components required for our device. The primary components of the acquisition card are a microprocessor and a PCI controller with bus master capability and support for DMA. The microprocessor is connected to ROM, SDRAM, an external momentary switch, an LED, and an external storage interface. The basic diagram is shown in Figure 2.

The on-board ROM contains the firmware operating code for the acquisition card. ROM is used because it is neither writeable nor field upgradeable. This prevents an unauthorized user from changing program code or operation or otherwise tampering with the card via the PCI bus. SDRAM is used to store variables, program stack, and other operating data. It is also used for the buffering and temporary storage of the volatile memory retrieved from the target PC. The LED provides device operation and status information to the user. For example, it could be used to identify the following states: Power On Self Test, Idle/Ready-to-Image, Imaging in Process, Error (of various types), Process Successful. The external switch is used to begin the imaging process and must be physically activated by a human. Although this original design assumes the server is in a physically secure operations center, future designs may assume a less secure environment. The external switch could be replaced by a secure user authentication device, such as a cryptographic smartcard, USB token, or one-time-password technology. Optional components include a real-time clock, used for accurate time-stamping to correlate actual time with host-assumed time, and secure Serial EEPROM memory, used to store user-configurable options (if applicable) and audit trail information.

In today's typical computer system, the PCI (Peripheral Component Interconnect) serves as a primary interface between the external adapter cards and internal system resources [16]. The North

Figure 2: Block diagram of the acquisition card hardware components



Bridge connects the host processor bus (to which the CPU and physical memory are attached) to the PCI bus. The North Bridge also allows devices on the PCI bus to access system memory at speeds approaching the target processor's full native bus speed. Figure 3 shows a basic layout of the PCI bus.

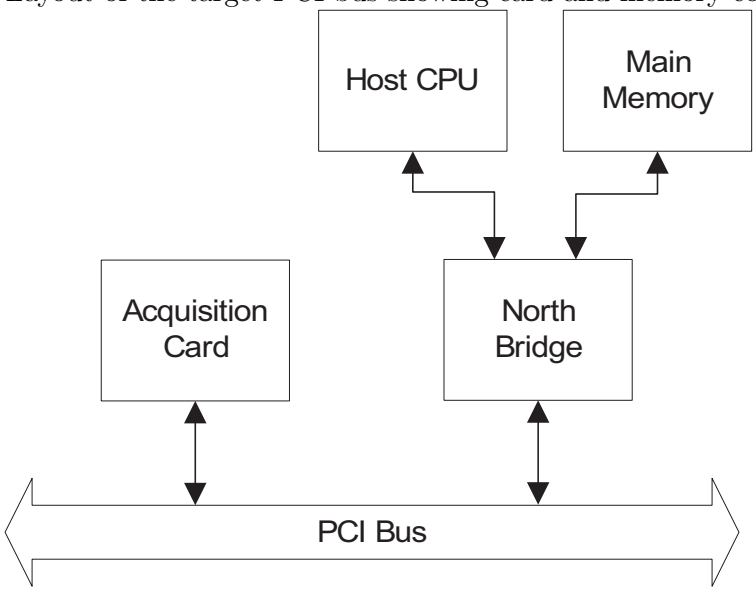
DMA is used to provide block transfers of data between the PCI bus and the target's local processor memory without requiring resources from the target processor itself. Using DMA allows a peripheral device, such as our acquisition card, to transfer data directly from memory without it being handled by the processor. During the acquisition process, our card will take control of the PCI bus and request a DMA transfer of system memory by specifying the desired base address and size of the block. The combination of PCI bus capabilities and DMA transfers are crucial to the effectiveness and reliability of our proposed acquisition card.

4.3 Imaging Procedure

We now provide a more formal description of our acquisition procedure. This includes the error handling procedures, but does not include implementation details, such as the hardware-specific steps.

1. The acquisition card is powered on and completes its hardware initialization routines.
   (a) The acquisition card conducts a Power-On Self Test (POST) and halts if a failure occurs.
   (b) The acquisition card does not enable its PCI controller.
   (c) The acquisition card remains idle until the external switch is activated.

Figure 3: Layout of the target PCI bus showing card and memory connections



2. When the external switch is activated, the acquisition process begins.

   (a) The acquisition card activates the external storage device and initializes it to store the upcoming memory image.

   (b) The acquisition card enables its PCI controller.

   (c) If possible, the acquisition card halts the target processor.

   (d) A log entry is created on the external storage device which states that an acquisition has started and includes the date and time, if available.

   (e) The acquisition card saves volatile system memory to the external non-volatile storage device by looping on the following steps, starting at physical memory address 0, until all memory has been read:

      i. If memory in the next `X` bytes of memory are known to be protected and not accessible, then read the unprotected areas using the following steps, write `'00'h` to the memory image corresponding to the protected areas, and create a log entry on the external storage device with the protected memory locations.

      ii. Perform a DMA Memory Read request for the next `X` bytes of memory to be written to a buffer in the acquisition card's SDRAM.

      iii. If an I/O error occurs, then a log entry is created on the external storage device and `X` bytes of `'00'h` are written to the memory image on the external storage device.

      iv. If no I/O error occurs, then the acquisition card writes the buffer in SDRAM to the memory image on the external storage device.

      v. If a hash value of the data is being calculated, then the `X` bytes of data are added to the hash calculation.

   (f) If a hash value of the data is being calculated, then the final hash value is added to the log or a separate file.

   (g) A log entry is created on the external storage device which states that the acquisition has ended and includes the date and time, if available.

(h) The acquisition card disables the PCI controller and deactivates the external storage device to end the acquisition process.

3. The acquisition card returns to an idle state.

The `X` bytes of memory that are read in the above loop will be dependent on the amount of memory on the acquisition card. We will need to conduct performance tests to identify a reasonable value for this. The external storage device should have a file system on it, such as FAT, so that multiple images may exist on the same storage device. Note that FAT32 file systems have a 2GB file size limit and may not be able to save the entire memory of large servers. Ideally, the memory image will be saved to a unique session directory with other files that provide optional system information.

5 Tribble: The Proof-of-Concept Device

To verify our procedure, we designed and built a proof-of-concept device called "Tribble," which is functional in our laboratory environment. Tribble is based on an Intel IQ80303 processor [11] which contains a 100MHz Intel 80960JT (i960) core, a glueless interface to ROM and SDRAM, an I2C bus interface, and a number of general purpose I/O pins. Most importantly, the IQ80303 processor (also referred to as the IOP303) contains a PCI-to-PCI bridge module, supporting primary and secondary PCI buses with up to a 528MBytes/second transfer rate for a 64-bit/66MHz PCI operation, and a DMA Controller, which handles the underlying timing and protocol specific functionality of DMA.

Using the Intel IQ80303K evaluation platform by Cyclone Microsystems provided us with a baseline design that only required minimal hardware modifications to fit our desired specifications. Tribble also contains 2MB of Flash ROM (programmed with the MON960 Target Monitor), 512MB of on-board SDRAM, RS232 UART (used as a "debug console"), two 7-segment LED displays, a Li-Ion backup battery (unused in our current design), and a momentary SPST pushbutton switch connected to the NMI (Non-Maskable Interrupt) of the IQ80303. A paper will be written in the future on the hardware implementation details.

Our development platform consists of a Dell Inspiron 8200 laptop running Windows 2000 with a 1.8MHz Intel Pentium 4M processor and 512MB of RAM. The Spectrum Digital SPI610 JTAG interface connects between the serial port of our development platform and the JTAG port of the Tribble device. Our software toolchain consists solely of the Intel i960 CTOOLS/MON960 package [10] which includes a C/C++ compiler, assembler, linker, runtime libraries, debugger, and target monitor. The `gdb960` debugger included in
CTOOLS allows for source-level debugging and the viewing and manipulation of memory and registers on the Tribble hardware (which was used extensively during operational testing and for the verification of successful volatile memory imaging).

Our target system is a Gateway desktop PC running Windows 2000 with a 1GHz Intel Pentium III processor and 256MB of RAM. The Tribble card is plugged into the PCI bus. Figure 4 shows a block diagram of our development environment.

The goal of this proof-of-concept device was to prove that system memory could be read via the PCI interface without modifying its contents. Due to the target monitor executing on the Tribble hardware, the PCI controller was enabled upon power up, so the hardware was visible to the system even when it was not acquiring memory. The external switch was not implemented in this design, so the acquisition process occurred programmatically when we downloaded code onto the Tribble hardware. Additionally, the external storage device interface was not implemented. The contents of retrieved volatile memory were viewed with `gdb960` and displayed to our serial debug console in both ASCII and hexadecimal formats, as shown in Figure 5.

Figure 4: Block diagram of the Tribble development environment

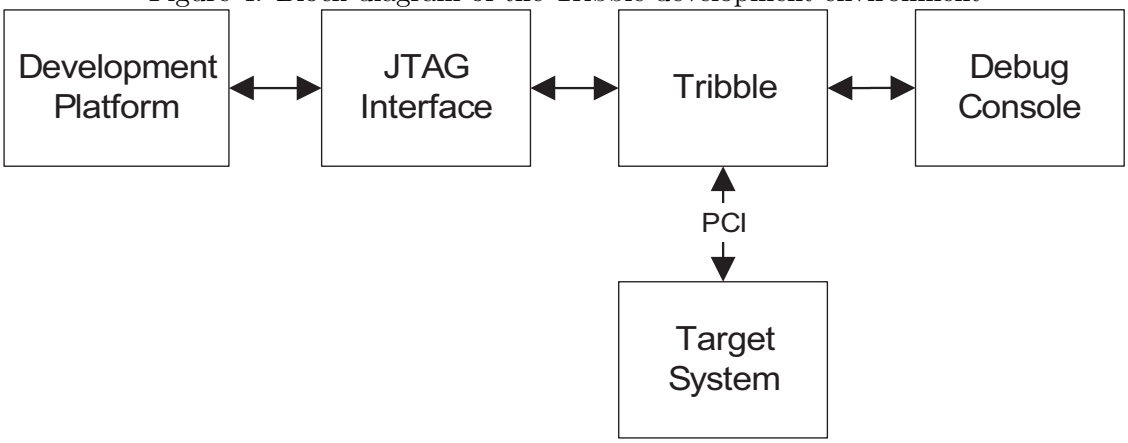


Figure 5: Partial memory dump displayed on the serial debug console, starting address 0x62EBBA0

```
040000000200000000E00C1000023100054005200490042004200 4C0045002000 [.............#..T.R.I.B.B.L.E. .]
570041005300200048004500520045002C002000440045004300 45004D004200 [W.A.S. .H.E.R.E.,. .D.E.C.E.M.B.]
450052002000310032002C002000320030003000330000000000 0000000000 [E.R. .1.2.,. .2.0.0.3..........]
000000000000000000000000000000000000000000004006A007 D020E0000103700 [.....................j.}.....7.]
780107007801070003C003D003E003F004000410042004300440045 0046004700 [x...x...<.=.>.?.@.A.B.C.D.E.F.G.]
480049004A004B004C004D004E004F005000510052005300540055 0056005700 [H.I.J.K.L.M.N.O.P.Q.R.S.T.U.V.W.]
580059005A005B005C005D005E005F006000610062006300640065 0066006700 [X.Y.Z.[.\.].^._.‘.a.b.c.d.e.f.g.]
680069006A006B006C006D006E006F007000710072007300740075 0076007700 [h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.]
780079007A007B007C007D007E007F00AC2081001A2092011E20262020202120 [x.y.z.{.|.}.˜.... ... ... &   ! ]
C602302060013920520 18D007D018F00900018201920 1C201D20222013201420 [..0 ‘.9 R...}..... . . . " . . ]
```

5.1 Test Procedures and Results

To test that Tribble was reading actual volatile memory using the PCI bus and DMA Read Memory requests, we configured it to read a 4096-byte page of memory and then read the same page of memory using `dd` with the
`\\.\PhysicalMemory` object. This was repeated with many pages and all were found to be equal, although many of the pages were just large arrays of `‘00’h`. We read data in 4096-byte blocks because that is the default size of virtual memory pages for Intel-based platforms.

To verify that both `dd` and Tribble were reading the correct memory locations, we used the SoftICE tool [4] to write two known 4-byte hexadecimal strings to two disparate physical memory locations. Both `dd` and Tribble read the page containing the memory locations that were modified by SoftICE and it was verified that the strings were there.

To test that the acquisition process was not modifying any memory that it read, a memory location with a known data value was read twice in a row. In all cases, the result of the second read was the same as the first. This test does not prove that other memory locations were not written to by Tribble during the process, but a more thorough test could not be performed until the memory contents are saved to a non-volatile external storage device.

A final test was to acquire all of the target system memory using the `dd` tool and then acquire all of the memory using Tribble. The output was then compared in 4096-byte pages. As previously mentioned, virtual memory uses pages that are 4096 bytes in size and therefore it would be expected that some pages would differ between the two acquisitions, since new pages may have been written

between the time of the acquisitions. Comparing parts of the two outputs by hand showed that many of the pages were the same, some of them had moved locations between the two executions, and some of the pages were new. Tribble ran very slowly due to the fact that the 256MB of memory was being written in ASCII and hexadecimal formats to a serial terminal at 115.2kbps, so many processes were able to run between the `dd` acquisition and the end of the Tribble acquisition. Therefore, these results were not surprising and more extensive tests will be performed when the external storage device interface is implemented.

Before the full memory acquisition was performed, the Notepad application was opened on the target machine and the string "TRIBBLE WAS HERE, DECEMBER 12, 2003" was entered into the window. This string was identified in Unicode format in both the Tribble and `dd` outputs. This string was previously shown in Figure 5.

During our full memory acquisition tests using Tribble, the target operating system would freeze when we attempted to read within memory location 0xB6000. This memory location is inside of the Upper Memory Area (UMA) of Intel-based PCs which is reserved for video RAM and system BIOS. The UMA ranges from 0xA0000 to 0xFFFFF (640kB to 1MB). For our tests, we skipped the memory region from 0xA0000 to 0xC0000 so that we would not freeze the target system. We will examine this memory area carefully in the future to identify any methods of reading it without causing ill effects on the target system. It may be necessary to skip the entire UMA region if other test systems freeze at different memory locations within the area.

6 Limitations

This section will address the limitations of this procedure. The first limitation of using the hardware-based acquisition device is that it needs to be installed prior to an incident. The device has not been designed for an incident response team member to carry in his toolkit to install after the fact, and rather needs to be considered as part of a forensic readiness plan.

The thought of installing a PCI card into multiple systems before an incident occurs may sound expensive and impractical. However, the intent of the card is not to be installed into every computer system in a particular environment. This device is most useful when installed into at-risk, critical servers where an attack is likely and a high-stake intrusion investigation might occur. From a cost perspective, the retail price of the hardware, though not fully explored at this point, will be equal to or less than other typical server components, making it both affordable and practical. Additionally, a trusted hardware-based device is needed to reliably aquire the system memory contents without compromising the integrity of data stored on the memory. Some servers have hot-swappable PCI card slots, so a responder could possibly install the card after an incident, but that will require access to the inside of the computer and may require use of the operating system in order to enable the device. This scenario and the use of other non-invasive interfaces (such as PCMCIA/CardBus) will be investigated in the future after the completion of our current design.

When the acquisition card enables its PCI controller, the operating system may detect that the card is now present on the bus. Systems with Plug-and-Play support, such as Microsoft Windows, may show a message that tells the user that a new device has been detected and may request a driver. To prevent this message, a simple "dummy" device driver could be installed on the system that is loaded when the card is enabled but does not actually interact with the physical card. The existence of this driver may alert the attacker that the card is installed, but due to the inherent operating system- and software-independence of PCI, we believe that he will not be able to disable or modify the behavior of the acquisition card. If it is determined through our future research that a device driver is indeed necessary, there are three possible scenarios if an attacker detects the presence of the driver. The attacker will either stop his attack altogether, continue with his attack

not concerned about the card, or attempt to attack the card itself. Future work will be done to verify that attacks against the card, such as through the creation of a malicious device driver, can be reduced or prevented.

The output of this process is a dump of physical memory. We will be able to do basic analysis of it using `strings`, `grep`, and hex editors, but there are currently no automated or integrated tools for analysis. To view the memory that a specific process used, the page table must be located and the pages in physical memory and swap space need to be mapped to the process's virtual memory addresses. This is also future work.

This type of device is difficult for an end user to test and validate. The memory is constantly changing and therefore it will be difficult to verify that the image is an exact copy and that data was not written to target memory during the acquisition. One potential test is to install the card in a computer with no operating system. The memory would be acquired twice in a row and the resulting images compared. When our implementation is complete, testing methods will be examined in more detail.

7 Conclusion

In this paper, we have given requirements for the general process of volatile memory acquisition, a procedure that can satisfy our requirements, and the initial results from our hardware implementation of the procedure. Our hardware-based procedure can provide more reliable evidence than a software-based solution because there is a smaller risk of an attacker modifying the procedure to produce false data. This method will also not write data to the target memory or hard drive during the acquisition.

A device that implements this procedure can be installed by a company on its servers so that the entire state of the server can be preserved for the search of digital evidence. This device can also be used by companies that do not have a dedicated incident response team. For example, a company detects that one of its systems has been compromised and a third party response team is called. However, it will take them at least an hour to arrive on site. During that time, the processes that the attacker has executed could cause damage to the system, but if the company unplugs the computer then the volatile memory is lost. Using the hardware-based implementation, the company can simply press a button to save the memory contents and then unplug the server until the response team arrives to acquire the disk and begin the investigation.

Our future work includes completing our hardware implementation of the procedure, conducting performance tests, verifying that system memory is not written to during the acquisition process, testing if the target system can be halted without ill effects, and investigating potential attacks against the device. A paper will be written in the future on the hardware implementation details. A patent for this device and procedure is pending.

References

[1] CAIDA. *Analysis of Code-Red*, July 2001. Available at: http://www.caida.org/analysis/security/code-red.

[2] Brian Carrier and Eugene Spafford. Getting Physical with the Digital Investigation Process. *International Journal of Digital Evidence*, Fall 2003.

[3] Eoghan Casey. Practical Approaches to Recovering Encrypted Digital Evidence. *International Journal of Digital Evidence*, Fall 2002.

[4] Compuware. *SoftICE*. Available at: http://www.compuware.com.

[5] Chris Drake and Kimberley Brown. *PANIC! UNIX System Crash Dump Analysis Handbook*. Pearson Education, 1995.

[6] Dan Farmer and Wietse Venema. *The Coroner's Toolkit (TCT)*. Available at: http://www.fish.com/tct.

[7] George Garner. *Forensic Acquisition Utilities*. Available at: http://users.erols.com/gmgarner/forensics.

[8] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. *Advanced Configuration and Power Interface Specification*. Available at: http://www.acpi.info/spec.htm.

[9] Hitachi. *Microdrive Overview Page*. Available at: http://www.hitachigst.com/hdd/micro/overvw.htm.

[10] Intel. *i960 CTOOLS*. Available at: http://developer.intel.com/design/i960/patches/index.htm.

[11] Intel. *IOP303 I/O Processor*. Available at: http://developer.intel.com/design/iio/80303.htm.

[12] ISS X-Force. *SQL Slammer worm propogation*, January 2003. Available at: http://xforce.iss.net/xforce/xfdb/11153.

[13] Timothy Lawless. *Linux Kernel-Level Trojan - Kernel Intrusion System (KIS)*. Available at: http://archives.neohapsis.com/archives/sf/linux/2001-q3/0038.html.

[14] National Institute of Standards and Technology. *Computer Forensic Tool Testing (CFTT) group*. Available at: http://www.cftt.nist.gov.

[15] National Institute of Standards and Technology. *Disk Imaging Tool Specification*, October 2001. Available at: http://www.cftt.nist.gov/DI-spec-3-1-6.doc.

[16] Tom Shanley and Don Anderson. *PCI System Architecture, Fourth Edition*. Pearson Education, 2000.

[17] Fred Smith and Rebecca Bace. *A Guide to Forensic Testimony*. Addison Wesley, 2003.

[18] Wietse Venema. *Forensic Discovery*. Available at: http://www.lysator.liu.se/upplysning/fa/wietse_susec.pdf.

[19] VMWare, Inc. *VMWare GSX Server*. Available at: http://www.vmware.com.