FireWire Memory Dump of a Windows XP Computer: A Forensic Approach

Antonio Martin
tmemail@gmail.com

Introduction

In a forensic investigation, while collecting evidence, Anzaldua, Godwin, Volonino state the best practice is to unplug a computer or remove a laptop's battery, so as to preserve the exact contents of a disk without introducing artifacts, changes to the system, for later investigations.[1] It is critical the original contents of a system, as found, are not altered. Using the operating system shut down alters log and temp file states; furthermore, a shutdown may trigger a logic bomb and a possible data wipe. This approach has a problem; it does not preserve active memory. A portion of the active memory can be found in the remnants of the operating system swap file, but this is an incomplete picture. There exists a desire to find a means to collect a forensic image of memory without compromising an investigation's integrity.

Network Based, Active Data Collection

As the field of digital forensics progresses, new means of evidence collection emerge. The current "best practice" allows for *In situ*, on sight, live data collection from running systems; a snapshot capture of all the information in memory. While desirable, it results in a series of problems that have not been addressed. Is this a case of law and forensic practitioners being technologically a few steps behind?

One example of the current means of network based, live data collection can be found in Guidance Software's Encase® Field Intelligence Model (EFIM); other software packages behave similarly. EFIM allows a forensics investigator to connect to a target machine by Ethernet, push a small program referred to as a "servlet" and capture the system's live memory and hard drive. EnCase FIM is capable of targetting "Windows 95/ 98/ NT/ 2000/ XP/ 2003 Server, Linux Kernel 2.4 and above, Solaris 8/9 both 32 & 64 bit, AIX, OSX."[2] At first glance, as long as the artifacts that are introduced by the "servlet" are well understood, the evidence collected should be trustable and admissible. The tool allows investigators to image running systems and servers while not disrupting a company's operations. Furthermore, it can be used to find and target machines on a network (wirelessly?), without the owner's knowledge.

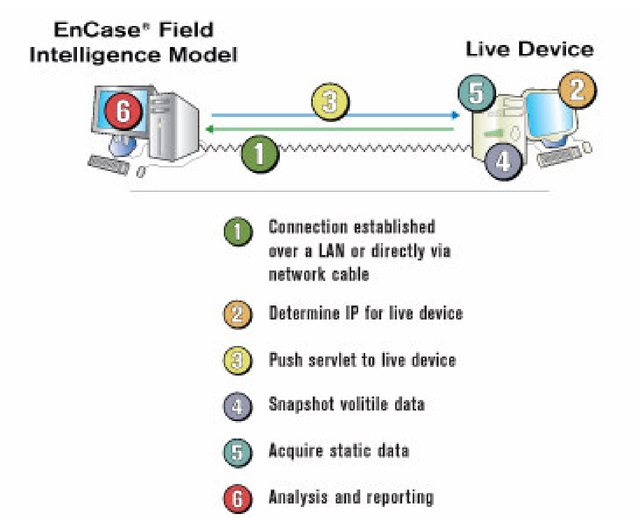

Figure 1. From www.guidancesoftware.com

EnCase's approach presents at least the following issues:

- A "servlet" installation introduces artifacts on the target system. Claims that the impacts are "well documented and known" are interesting; it is impossible to test and document all possible hardware and software configurations and how their interactions will affect the impacts of installation and operation.
- A "servlet" is susceptible to attack since malicious software running on the target can identify and stop it or trigger a logic bomb. This becomes more of an issue since the full Encase product suite is available from torrents and warez sites allowing criminals to dissect and build defenses. While the "servlets" can be updated and altered, it will require investigators to constantly update and document changes to their investigation test stands and run the risk of loosing all data if a logic bomb is triggered. Simple file

modification of the “servlet” might not work to prevent detections as data stream and behavior analysis can quickly identify potential alterations/updates.

- It is possible the “servlet” could be maliciously submitted to malware and virus protection houses. The code would be inspected and signature detection profiles pushed out to millions of computers world wide. Thus a target with active, running virus protection might automatically stop a live forensics investigation.
- By far, the most critical issue: Any targeted system is an unknown quantity with little insight as to what is or is not running. If a system is infected, it is impossible to trust any information gathered through the operating system because stealth root kits are difficult, if not impossible to detect. Rootkits have advanced considerably and data from an unknown, possibly infected system cannot be trusted. [3, 4] A kernel mode rootkit called “Shadow Walker,” available since summer of 05, is such a program; it hides itself and other processes within memory by subverting the windows virtual memory management. Any process from within the CPU attempting to view memory will be fooled.

> *”Unfortunately, because all live response tools of which we are aware run directly on a potentially compromised system, they rely on the underlying operating system, …. Even tools which attempt to determine the integrity of the operating system may be fooled if the attacker has perfect knowledge of the tool and control of the system before it is installed. … it is impossible to even know if the live response tool itself has been run in an unmodified way. This means that, even if the tool itself has been verified, the executing instance of that tool may be untrustworthy.”* [Butler, Sparks 4]

It is possible that a rootkit could be discovered on analysis of the hard drive copy, if the rootkit allowed the drive section to be read. The rootkit might not exist in the drive, but could hide its image in the BIOS or a video card ROM.[3]

The process and end product of gathering forensic evidence is supposed to be of the utmost quality and integrity, but this approach to live evidence collection is flawed and approaches negligence if it makes such claims. Ways to subvert and bypass such methods are already used and well documented. While this can be a useful tool for information gathering, this form of forensic evidence collection should not be admissible in a court of law where integrity of data is of the utmost concern because it cannot be guaranteed.

Firewire 1394 or or

Firewire® is a bus technology designed for point to point connections between devices. It was developed by Apple in 1986 and was standardized to the IEEE 1394 specification in 1995.[5] Firewire, like many other devices in today's computer architecture, utilizes Direct Memory Access (DMA) to improve data transfers.

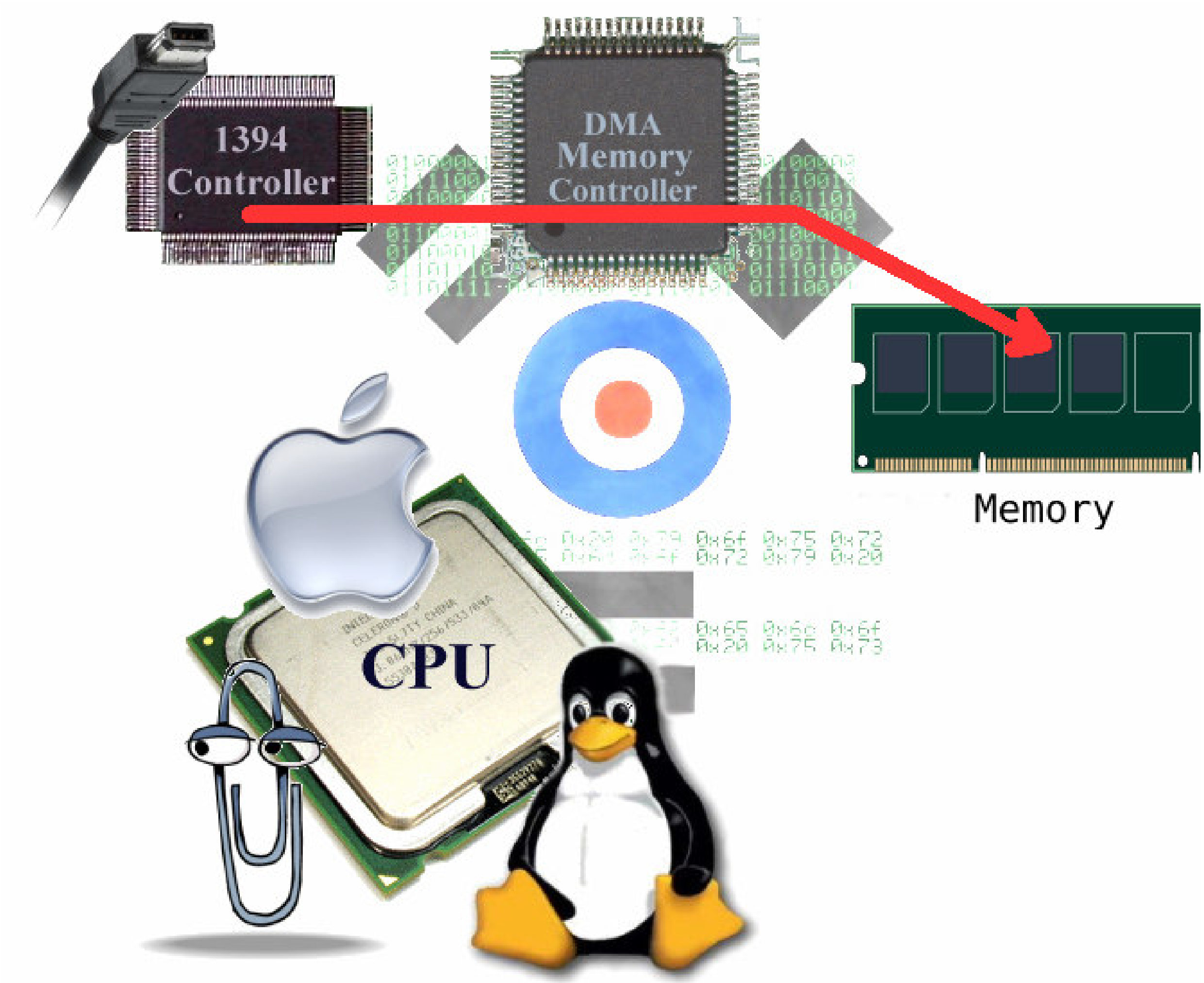

Figure 2: Firewire, using DMA, bypasses the CPU and running operating system.

A firewire device can read (and write) to a computer's main memory by accessing a system's DMA controller, while the operating system, be it Windows, Mac OS, Linux, a Multiple Independent Levels of Security kernel, etc., is oblivious to the event. By pulling a copy of memory through firewire, the target CPU and operating system are bypassed as are any infections, triggers or traps. This is not a bug but exactly how DMA and PCI devices, like Firewire, were designed to operate.

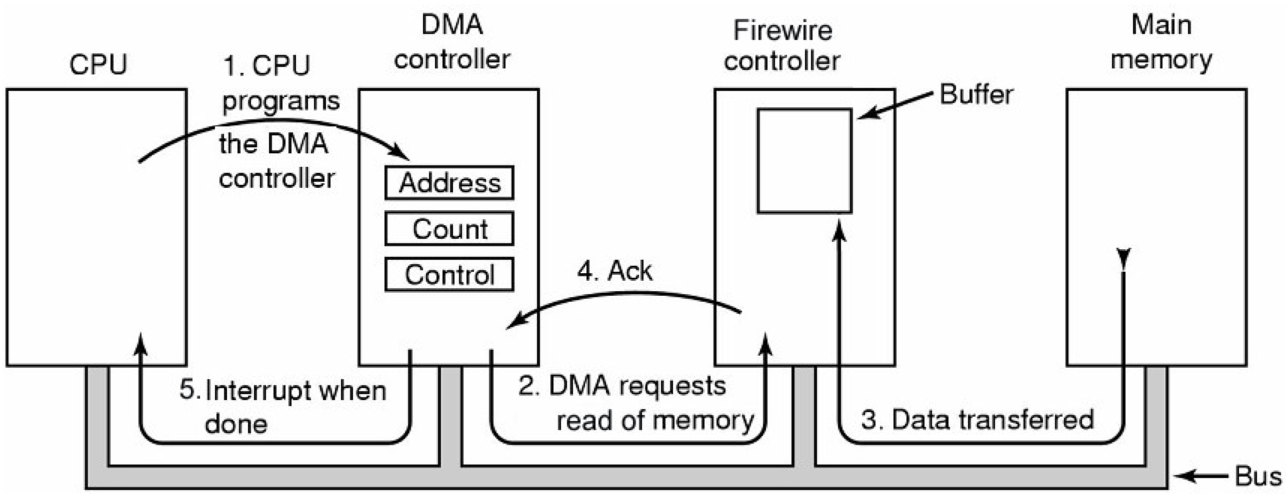

Figure 3: Modified from IO class notes, Dr Szabolcs Mikulas, School of Computer Science and Information Systems, Birkbeck College, Chapter 5, Input / Output [6]

DMA allows memory transfers between devices and processes to take place while a computer's CPU performs other tasks. Figure 3:

1. The CPU/operating system programs the DMA controller to instruct a Firewire device to read a portion of memory; the CPU/operating system is now free to work on other tasks.
2. The DMA controller sends a message to the Firewire controller, informing it of the read request and the location and length in memory.
3. The Firewire device negotiate control of the PCI bus and reads the memory location specified and once completed,
4. Informs the DMA controller.
5. Finally, the DMA controller triggers an interrupt, informing the CPU the read has completed.

It should be noted that devices are not limited to only reading/writing to the memory address specified by the operating system. Firewire and other DMA bus master devices act independently of the CPU; the CPU need not initiate the transaction. A firewire device can program the DMA controller and set up its own reads and writes, as per the PCI and IEEE 1394 specifications.

"Hit by a Bus: Physical Access Attacks with Firewire"

At PacSec in November of 2005, Maximillian Dornsief presented a paper, "*0wned by an iPod*" where he demonstrated how a firewire device, utilizing DMA, can read/write active memory within a Mac, BSD or Linux machine.[7] At Ruxcon 2006, Adam Boileau (aka Metlstorm) from Security-Assessment.com presented "*Hit By A Bus: Physical Access Attacks with Firewire*" where he extended prior works, enabling the targeting of a Windows XP machine. Utilizing a Linux box with firewire support, and a set of tools for enabling the interface, he revealed several hacks using live memory reading and writing targeted against Windows XP.

- Reading over firewire the entirety of the target's memory and saving it to disk without altering the target's state.
- Locating and over writing a memory address containing the graphical identification and authentication library (msgina.dll) allowing the password on a locked Windows XP machine to be bypassed.
- Pushing to the target computer and starting a process without the process existing on the target hard drive.
- Recovering the last sixteen bytes from the keyboard buffer accepted by the BIOS prior to booting the primary operating system, useful in finding BIOS and disk encryption passwords.

Along with the presentation was released a set of Linux tools for firewire memory reading and writing. [8]

What is possible?

Utilizing MetlStorm's toolset, it is possible to configure a Linux system (in this case Ubuntu 6.10) and target an IBM Trusted Computing Module enabled Thinkpad.

```
00000600  FA 33 C0 8E C0 8E D8 8E  D0 BC 00 7C 8B F4 FB FC  ú3À▮À▮Ø▮Ð¼.|▮ôûü
00000610  BF 00 06 B9 00 01 F3 A5  B8 DF 06 50 C3 00 0F 00  ¿..¹..ó¥¸ß.PÃ...
00000620  01 0A 45 72 72 6F 72 20  6C 6F 61 64 69 6E 67 20  ..Error loading 
00000630  6F 70 65 72 61 74 69 6E  67 20 73 79 73 74 65 6D  operating system
00000640  0A 0D 00 0A 49 6E 76 61  6C 69 64 20 70 61 72 74  ....Invalid part
00000650  69 74 69 6F 6E 20 74 61  62 6C 65 0A 0D 00 50 72  ition table...Pr
00000660  65 73 73 20 6B 65 79 20  74 6F 20 72 65 62 6F 6F  ess key to reboo
00000670  74 20 0A 0D 00 80 7C 04  0C 74 1A 80 7C 04 0E 74  t ...▮|..t.▮|..t
00000680  14 81 7C 0A FA 00 73 0D  8B 4C 02 8B 14 B2 80 B8  .▮|.ú.s.▮L.▮.²▮¸
00000690  01 02 CD 13 C3 56 8B C3  87 DE BE 00 06 C7 04 10  ..Í.ÃV▮Ã▮Þ¾..Ç..
000006A0  00 C7 44 02 01 00 89 44  04 8C 5C 06 8B 47 08 89  .ÇD...▮D.▮\.▮G.▮
000006B0  44 08 8B 47 0A 89 44 0A  C7 44 0C 00 00 C7 44 0E  D.▮G.▮D.ÇD...ÇD.
```

Figure 4: Memory dump containing a system's BIOS.

The firewire dump produces a large (the size of available memory) and difficult to decipher binary image. Figure 4 shows a portion of the capture displaying the target's BIOS in memory. This file is difficult to understand without in-depth knowledge of the target operating system's memory map; thankfully there are several tools available to assist. Andreas Schuster created Process and Thread Finder (PTFinder), a script capable of parsing firewire memory dumps.[9] With this script and a little work, the following information can be gathered:

An investigator can quickly know what programs are/were running and can view the various program/thread memories for information about activity, connections, username password combinations, etc. Below is an example of a PTFinder list from the IBM memory dump. It dumps all threads (and processes), their thread IDs, associated Process IDs, times created, exited, offset/location within the memory dump (so you can go to the thread's location and view the information), the PDB (processes virtual address value) and Remarks (usually the process name or system's status).

```
No.  Type PID   TID   Time created        Time exited         Offset     PDB        Remarks
---- ---- ----- ----- ------------------- ------------------- ---------- ---------- ---
   1 Thrd     0     0                                         0x00559320
   2 Proc     0                                               0x00559580 0x00039000 Idle
   3 Thrd     4  3284                     2006-08-03 09:12:35 0x02a66da8
   4 Thrd     4  3496                     2006-08-03 16:43:45 0x02a80da8
   5 Thrd     4  4048                     2006-08-03 16:43:45 0x02a81da8
   6 Thrd     4  3088                     2006-08-03 16:43:42 0x02a82da8
   7 Thrd     4  3412                     2006-08-03 16:43:42 0x02a83020
   8 Thrd     4  3572                     2006-08-03 16:43:42 0x02a833c8
   9 Thrd     4  2660                     2006-08-03 16:43:42 0x02a83640
  10 Thrd     4  4036                     2006-08-03 16:43:42 0x02a838b8

...
 649 Proc  2368                           2006-07-31 16:18:15 0x0368ba98 0x16ef8000 ibmmessages.exe
 695 Proc     4                                               0x037c87c0 0x00039000 System
```

Running a grep on the list or using a flag for PTFinder can produce a list of just the processes running that were found in the memory dump. Note the first two, red highlighted lines (No. 55 and 247), a Back Orifice 2k configuration tool (bo2kcfg.exe) and the BO2K GUI (bo2kgui.exe) used to control BO2K infected machines. In some cases, the residual memory from prior processes and threads is still available; the third highlighted line (processes SynTPLre.exe, No. 292) shows a recently exited process whose memory is still available.

```
No.  Type PID  TID  Time created        Time exited         Offset     PDB        Remarks
---- ---- ---- ---- ------------------- ------------------- ---------- ---------- ---
   2 Proc    0                                              0x00559580 0x00039000 Idle
  55 Proc 2060      2006-08-03 16:32:39                     0x02d8d020 0x24445000 bo2kcfg.exe
 148 Proc 3388      2006-08-03 16:51:05                     0x02e1c4c0 0x28659000 msmsgs.exe
 247 Proc  624      2006-08-03 16:32:41                     0x02ee0020 0x17a4f000 bo2kgui.exe
 254 Proc 2392      2006-07-31 16:18:17                     0x02eedda0 0x16c57000 tfswctrl.exe
 292 Proc 2156      2006-07-31 16:18:11 2006-07-31 16:18:12 0x02f2fbc0 0x14baf000 SynTPLpr.exe
 293 Proc 2056      2006-07-31 16:18:09                     0x02f31800 0x1499e000 TpScrex.exe
 295 Proc 1972      2006-07-31 16:18:08                     0x02f33da0 0x14597000 TPONSCR.exe
 297 Proc 2480      2006-07-31 16:18:20                     0x02f38638 0x1755d000 certtool.exe
 299 Proc 2528      2006-07-31 16:18:22                     0x02f3bda0 0x1789e000 pwmgr.exe
 308 Proc  724      2006-07-31 16:18:06                     0x02f4c608 0x1408c000 QCWLICON.EXE
 321 Proc  968      2006-07-31 16:18:04                     0x02f59620 0x13d54000 TPHKMGR.exe
 329 Proc 1804      2006-07-31 16:17:53                     0x02f619e0 0x1323a000 explorer.exe
 344 Proc 2588      2006-07-31 16:18:22                     0x02f77488 0x17ead000 ccApp.exe
 349 Proc  224      2006-07-31 16:15:10                     0x02f7d630 0x0e557000 alg.exe
 362 Proc 1904      2006-07-31 16:15:08                     0x02f8ada0 0x0db41000 SymWSC.exe
 371 Proc 1844      2006-07-31 16:15:07                     0x02f96da0 0x0d939000 TpKmpSvc.exe
 389 Proc 1708      2006-07-31 16:15:06                     0x02faf020 0x0d129000 RegSrvc.exe
 390 Proc 1744      2006-07-31 16:15:07                     0x02faf800 0x0d22e000 UMGR32.EXE
 393 Proc 1688      2006-07-31 16:15:06                     0x02fb4800 0x0d10b000 TssCore.exe
 406 Proc 1636      2006-07-31 16:15:06                     0x02fc8da0 0x0d122000 QCONSVC.EXE
 419 Proc  896      2006-07-31 16:18:02                     0x02fd8948 0x14116000 SynTPEnh.exe
 427 Proc 1592      2006-07-31 16:15:06                     0x02fe2800 0x0cc1b000 uvmserv.exe
 430 Proc 1544      2006-07-31 16:15:06                     0x02fe3bc0 0x0ce16000 ati2evxx.exe
 439 Proc 1440      2006-07-31 16:15:05                     0x02feebc0 0x0cc08000 spoolsv.exe
 463 Proc 1308      2006-07-31 16:15:05                     0x03003b88 0x0c1fe000 ccEvtMgr.exe
 479 Proc 1764      2006-07-31 16:18:05                     0x0300cd40 0x13b76000 TP98TRAY.EXE
 485 Proc 1128      2006-07-31 16:15:04                     0x03011da0 0x0c0f6000 svchost.exe
 492 Proc  976      2006-07-31 16:15:03                     0x03018800 0x0bbd2000 svchost.exe
```

```
494 Proc  492      2006-07-31 16:18:02                    0x0301ada0 0x13dd0000 SynTPLpr.exe
519 Proc 1084      2006-07-31 16:15:04                    0x032098b0 0x0c0f0000 svchost.exe
523 Proc  876      2006-07-31 16:15:01                    0x0320c648 0x0b79d000 svchost.exe
535 Proc 1032      2006-07-31 16:15:04                    0x0321f020 0x0bfe5000 S24EvMon.exe
557 Proc  840      2006-07-31 16:14:58                    0x03380da0 0x0b696000 ibmpmsvc.exe
563 Proc  656      2006-07-31 16:14:57                    0x033b0bc0 0x0aae0000 lsass.exe
579 Proc  644      2006-07-31 16:14:57                    0x033f29c0 0x0a9d6000 services.exe
582 Proc  600      2006-07-31 16:14:55                    0x03400da0 0x0a9b4000 winlogon.exe
608 Proc  556      2006-07-31 16:14:48                    0x034f9da0 0x09dae000 csrss.exe
621 Proc  940      2006-07-31 16:15:02                    0x03559600 0x0b9cb000 svchost.exe
637 Proc  500      2006-07-31 16:14:40                    0x035b23e0 0x08943000 smse.exe
644 Proc 2188      2006-07-31 16:18:12                    0x03680da0 0x15bfc000 AGRSMMSG.exe
646 Proc 2348      2006-07-31 16:18:15                    0x03688aa8 0x164c3000 tgcmd.exe
647 Proc 2176      2006-07-31 16:18:12                    0x036894f8 0x14976000 EzEjMnAp.Exe
649 Proc 2368      2006-07-31 16:18:15                    0x0368ba98 0x16ef8000 ibmmessages.exe
695 Proc    4                                             0x037c87c0 0x00039000 System
```

Six hundred and ninety five threads were identified and notated in the memory dump by PTFinder. A useful enhancement to the PTFinder tool would be the ability to save the memory sections for the individual processes and associated threads, each stored in their own file grouped by directory. This would allow quicker and easier examination and categorization.

```
02D4C720  4C 4D 45 4D 28 00 00 00  FC 67 17 00 78 36 16 00  LMEM(...üg..x6..
02D4C730  47 00 08 00 B1 01 0A 00  A8 19 18 00 6F 72 69 7A  G...±...¨...oriz
02D4C740  61 74 69 6F 6E 3A 20 50  61 73 73 70 6F 72 74 31  ation: Passport1
02D4C750  2E 34 20 4F 72 67 56 65  72 62 3D 47 45 54 2C 4F  .4 OrgVerb=GET,O
02D4C760  72 67 55 52 4C 3D 68 74  74 70 25 33 41 25 32 46  rgURL=http%3A%2F
02D4C770  25 32 46 6D 65 73 73 65  6E 67 65 72 25 32 45 6D  %2Fmessenger%2Em
02D4C780  73 6E 25 32 45 63 6F 6D  2C 73 69 67 6E 2D 69 6E  sn%2Ecom,sign-in
02D4C790  3D 6E 6F 61 63 63 6F 75  6E 74 31 32 33 34 25 34  =noaccount1234%4
02D4C7A0  30 68 6F 74 6D 61 69 6C  2E 63 6F 6D 2C 70 77 64  0hotmail.com,pwd
02D4C7B0  3D 70 61 73 73 77 72 64  2C 6C 63 3D 33 33 33 33  =passwrd,lc=3333
02D4C7C0  2C 69 64 3D 33 33 33 2C  74 77 3D 33 33 2C 72 75  ,id=333,tw=33,ru
02D4C7D0  3D 68 74 74 70 25 33 41  25 32 46 25 32 46 6D 65  =http%3A%2F%2Fme
02D4C7E0  73 73 65 6E 67 65 72 25  32 45 6D 73 6E 25 32 45  ssenger%2Emsn%2E
02D4C7F0  63 6F 6D 2C 63 74 3D 31  31 37 35 35 31 39 39 38  com,ct=117551998
02D4C800  37 2C 6B 70 70 3D 31 2C  6B 76 3D 39 2C 76 65 72  7,kpp=1,kv=9,ver
02D4C810  3D 32 2E 31 2E 36 30 30  30 2E 31 2C 72 6E 3D 61  =2.1.6000.1,rn=a
```

Figure 5: Memory section for Windows Messenger with username and password to the account.

Opening the saved memory image file in a hex editor (WinHex used) allow an examiner to find the memory sections pointed to by the PTFinder dump. Referencing the Windows Messenger (msmsg.exe) process id 3388, it is possible to find all associated threads in the dump and go to those memory offsets. Figure 5 shows an example thread 2248's memory section that contains the Windows Messenger's sign-in id and password.

```
17B7AC00  00 00 00 00 00 00 00 00  00 00 00 00 0C 00 00 00  ................
17B7AC10  0C 00 00 00 4C 4F 47 4F  4E 5F 4F 42 4A 45 43 54  ....LOGON_OBJECT
17B7AC20  30 1C 3E 02 05 00 00 00  00 00 00 00 05 00 00 00  0.>.............
17B7AC30  44 61 76 65 00 00 46 00  00 00 00 00 0E 00 00 00  Dave..F.........
17B7AC40  53 59 53 54 45 4D 5F 55  4E 4C 4F 43 4B 00 46 00  SYSTEM_UNLOCK.F.
17B7AC50  0C 01 00 00 5C 1C 3E 02  0C 01 00 00 03 00 00 00  ....\.>.........
17B7AC60  50 61 73 73 77 6F 72 64  31 31 00 00 00 00 00 00  Password11......
17B7AC70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```
Figure 6: Memory dump from an IBM Thinkpad with TMP, user ID and password readable.

Looking into kernel memory can reveal interesting information. Figure 6 displays a portion of memory taken from an IBM (now Lenovo) Thinkpad with a Trusted Computing Module and associated software. A TMP laptop is marketed offering a higher level of security over a standard laptop by leveraging TMP protections. This structure was found in high memory and contains the username, Dave, and password, Password11, of the person currently logged into the targeted machine. While a TPM system is supposed to operate at a higher level of security, the reality is much different.

The set of firewire tools for creating the memory images from MetlStorm has been added to the FCCU GNU/Linux Forensic Boot CD and can be found at http://www.lnx4n6.be/. [10]

Konuku's Volatools also offers a set of tools for analyzing memory images but it appears to not be designed for firewire dumps.[11] It failed on many of the attempts to parse most information from the file, like processes and threads but was able to find the current computer time.

```
E:\Python25>   python volatools ident -f memoryimage.bin
              Image Name: memoryimage.bin
              Image Type: XP SP2
                 VM Type: nopae
                     DTB: 0x39000
                Datetime: Thu Aug 03 09:51:16 2006
```

It also has support to find open sockets and network connection addresses (also failed). A potentially useful tool in analyzing Windows memory dumps.

Can it be defeated?

At the February 2007 BlackHat convention in DC, Joanna Rutkowska demonstrated how to defeat DMA based memory gathering by utilizing a low level program, in the CPU, to rewrite a computer's North Bridge memory lookup table's pointers.[12] The north bridge hosts its own memory lookup table (IO Memory Management Unit), a mapping of the addresses and layout in main memory. By rewriting the lookup table, three possible means of preventing firewire/DMA based memory reads can be realized. The first redirects the IO back at the system bus; this causes the computer to freeze, probably because the address range was not valid. The second redirects the IO so that all data returned is 0xff. Lastly, a more stealth means where certain memory pages can be hidden by removing their pointers and addressing them to other locations.

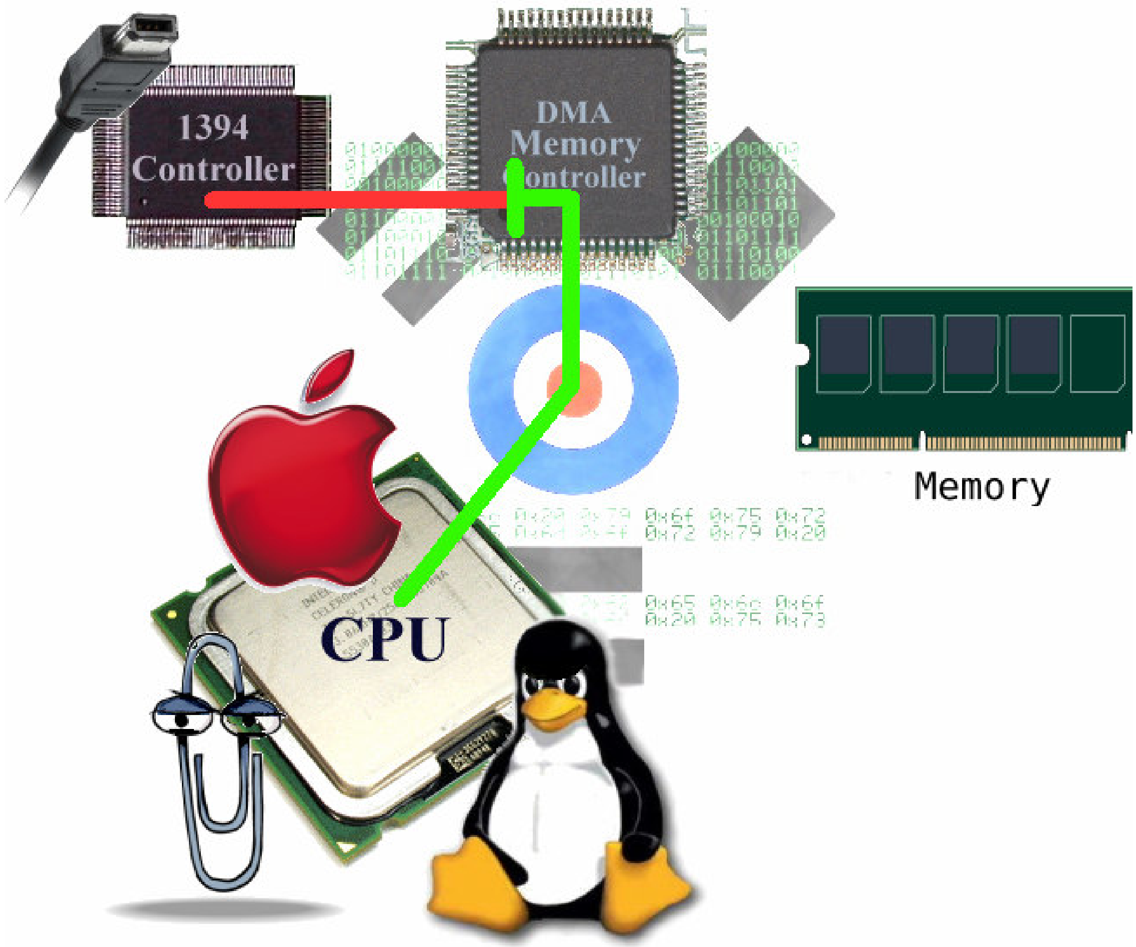

Figure 7: The operating systems get mean, remapping the DMA address table, preventing DMA.

Imaging a Live Drive by Firewire?

Apple's computers offer a "Target Disk Mode," providing the ability for one Mac computer to boot off of another's drive by a firewire connection. This leads to the possibility of not only collecting active memory from a running system but also the contents of the hard drive.

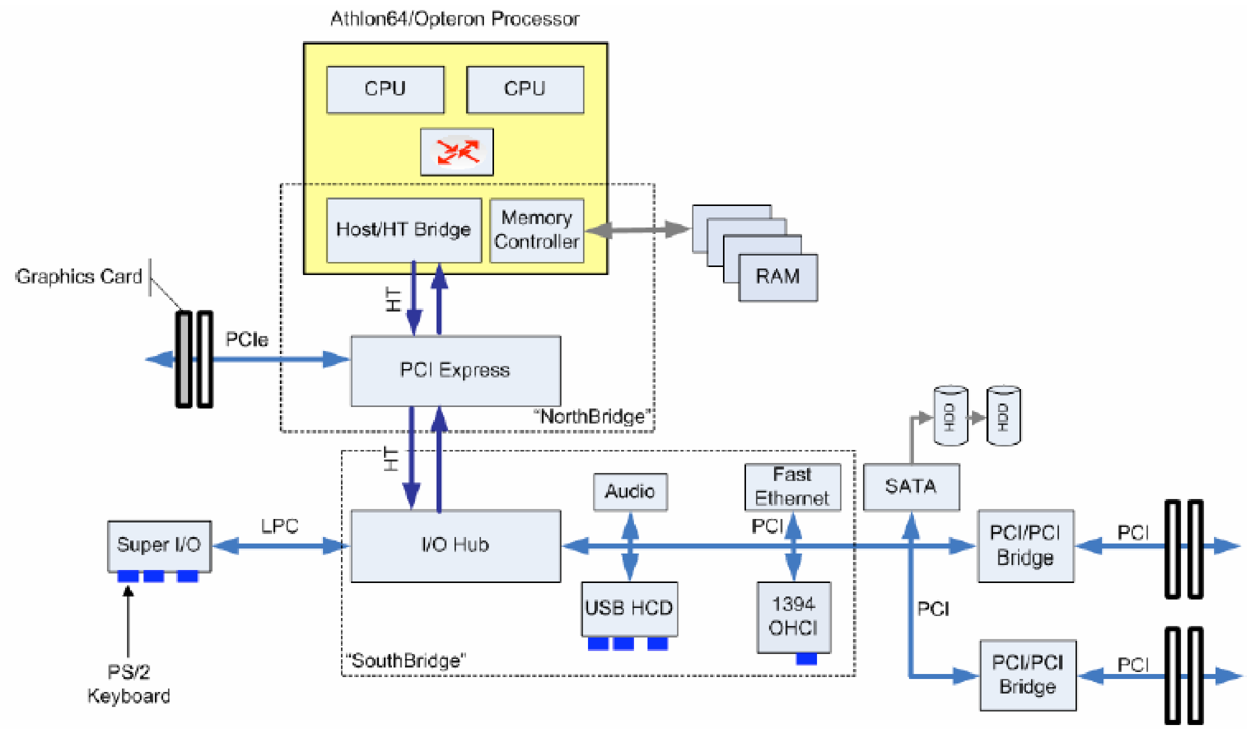

Figure 8: © COSEINC Advanced Malware Labs [x]Device and bus layout for an AMD computer. [12]

Since Firewire can access memory, it should also be possible for it to access other devices through the DMA controller. This would require a transfer through memory and could possibly be detected by the CPU since memory would be altered. Joanna Rutkowska demonstrated how the CPU can remap the memory map in a DMA controller; specifically one of the cases redirected the firewire's (PCI device) requests back towards the PCI bus, causing the system to halt. Given a PCI device can alter these mappings to read memory, it should be possible to find the IO device mapping and remap the DMA controller back at a hard drive's ports. Since the firewire and Serial ATA drives both sit on the same PCI bus, the firewire device might be able to directly access the drive, bypassing the need for using DMA. Current video cards are capable of transferring data directly to each other on a PCI(Express) bus, bypassing the need to communicate back to the CPU, is it a big step to the directly address a hard drive? Could this be another possibly powerful tool in the hands of the forensics investigator?

Conclusion

Firewire collection presents a few problems:

- Limited availability of firewire ports on computer systems.

- Plugging in a firewire device might require the operating system to activate the port, a slight alteration (artifact) to the state of the system.
- It is possible to crash a system if not done properly; at least the hard drive state would be preserved.
- Primarily, the concepts and tools availability for firewire memory imaging are still immature.

Active memory and live data collection are new to the field of digital forensics and still present a multitude of issues. The desire to collect a snapshot of what is happening in a system is of great value but this must not override the greater value of preserving the integrity of the data collected.

References

[1] R. Anzaldua, J. Godwin, L. Volonino, "*Computer Forensics Principles and Practices*" Prentice Hall, 2006

[2] "*Field Intelligence Model Detailed Product Description*", Guidance Software, 2006 http://www.guidancesoftware.com/downloads/getpdf.aspx?fl=.pdf

[3] A. Martin, "*Viral Threats - An Examination of Current and Evolving Technologies.*" Boston University Conference on Information Assurance and Cyber Security, Dec 2006

[4] J. Butler and S. Sparks, "*Windows rootkits of 2005, part two*" SecurityFocus, Nov 2005, http://www.securityfocus.com/infocus/1851

[5] "*Hardware & Drivers – FireWire*" Developer Connection, Apple Inc., 2007 http://developer.apple.com/hardwaredrivers/firewire/index.html

[6] S. Mikulas Ph.D., "*Chapter 5 – Input / Output*", Class Notes, School of Computer Science and Information Systems, Birkbeck College, 2006

[7] M. Dornseif, "0wned by an iPod" Laboratory for Dependable Distributed Systems, PacSec 2004 http://md.hudora.de/presentations/firewire/PacSec2004.pdf

[8] A. Boileau (aka Metlstorm) "*Hit By A Bus: Physical Access Attacks with Firewire*" Security-Assessment.com, Ruxcon 2006 http://www.storm.net.nz/static/files/ab_firewire_rux2k6-final.pdf

[9] A. Schuster, Process and Thread Finder. International Forensics Blog, forensikblog.de 2006

[10] C. Monniez, lnx4n6 Forensic Boot CD. The Belgian Computer Forensic Website http://www.lnx4n6.be/

[11] N. Pertroni Jr., A Walters, "*Volatools: Integrating Volatile Memory Forensics into the Digital Investigation Process*" Komoku.com 2007 http://www.komoku.com/pykvm/basic/bh-fed-07-walters-paper.pdf

[12] J. Rutkowska, "*Beyond The CPU: Defeating Hardware Based RAM Acquisition Tools*" COSEINC Advance Malware Labs, Black Hat DC 2007, Febuary 2007

1394 Open HCI, "1394 Open Host Controller Interface Specification" Release 1.1, Jan 2000

D. Anderson, T. Shanley, "PCI System Architecture" Addison-Wesley, Sept 2003