Probing System Hardware in Linux

by David Bank RHCT, CNE, CCSE, CCNA

v1.00 (2011-Jan-14)

Scope and Focus

The topic of this paper is OS-level coding (using shell scripts and Linux-typical executables) to analyze and quantify hardware (whether physical or virtual) underlying a Linux server (*e.g.* hardware common to a data center, as opposed to desktops). Where specific hardware vendors are mentioned, it is for the purposes of demonstrating a specific technique – it is not possible for this paper to delve into even a fraction of the various hardware OEMs, nor can every potential technique or tool be demonstrated. As exemplars, this paper uses commands on a limited number of models of Gateway/eMachine, HP, IBM and Sun hardware for physical platforms, and KVM and VMware for virtual platforms.

This paper will focus on the **x86** architecture, and will also assume a GNU/Linux distribution that is a RedHat variant (for example, Centos) or close cousin. Other distributions may or may not not have some or all of the tools mentioned, or the tools may exist under different names. Finally, this paper is geared towards both BASH and a 2.6 kernel environment. As a baseline, consider this paper applicable to RHEL v4.0 Update 8 or later.

Detection *vs.* Analysis

Detection of hardware – that is, determining if a specific bit of hardware exists within the system – is often easier than analysis of the hardware. For example, counting Ethernet NICs in a host is simple; determining the operational attributes of each NIC (*e.g.* MAC address, link state, *etc.*) takes considerably more effort.

This paper will provide tips and techniques for both tasks (detection and analysis), but those techniques should be tested in the reader's specific environment. Results may vary between hardware platforms (due to chipset variations) or even OS releases (as a result of driver versions, kernel changes and/or variations in tools).

Assumptions and Requirements

In order to fully leverage the tools and techniques in this paper, you must have an operational x86 host with a working Linux distribution already installed (or perhaps run from LiveCD). Additionally, you must have privileged access to the host.

Finally, since the paper approaches the topic from the OS perspective, it is assumed that the OS supports, has properly detected, and has loaded the correct driver(s) for the hardware in the system. The tools and techniques presented are not designed to sleuth out hardware that the OS itself has not found or does not support[1].

Platform

One of the most useful initial tasks for analyzing hardware is determination of whether or not a host is virtual; and also to determine the vendor, or OEM, of the host. When looking for a starting point[2], perhaps the best, but hardly only, tool for this is **dmidecode**[3], which is found in practically all Linux distributions. Note that older versions of this tool, while outputting much the same information as newer versions, do not support the **-t** parameter; this paper assumes you're using at least v2.8.

Start your hardware analysis with the command:

```
dmidecode -t 1 | grep Manufacturer | awk '{print $2}'
```

The output will be a string of variable length. The following table will help you interpret the values:

String	Notes
Gateway	Seen on eMachine hardware
HP	Seems to correspond to an HP/Compaq “server class” machine
Hewlett-Packard	Similar to **HP**, but seems to be used for “desktop class” machines
IBM	All IBM systems, from laptops to servers
Red	Probably RedHat-delivered QEMU KVM; you can confirm this with the command `dmidecode -t 0 | grep Vendor | awk '{print $2}'`, which should print the string **QEMU**
Sun	Indicates Sun (now Oracle) x86 hardware
VMware,	Note the trailing comma! This indicates a VMware guest machine

As part of interpretation, you can also see which ones are virtual platforms. Note that only a few examples are shown – many values are possible, but it is not within the scope of this paper to provide a definitive guide to the strings associated with each and every OEM.

Physical Host Model and Blade Slot

For physical hosts, you can usually determine the OEM's model number; if the host is on Blade hardware, the chassis slot number may also be accessible to you.

Compaq/HP Hosts: Model and Blade Slot

On HP hosts, start with:

```
dmidecode -t 1 | grep Product | awk '{print substr($4,1,2)}'
```

If the result is **Co** (the full string will be **Compaq**), then this will confirm that the underlying hardware is an HP “desktop class” system (such as a ProSignia). You can retrieve the model designation using:

```
dmidecode -t 1 | grep Product | awk '{print $5}'
```

Otherwise, the result should be either **BL** (indicating a Blade-based host), or **DL** (for a rack-mounted system). In either case, you can refine the statement to get the full model number:

```
dmidecode -t 1 | grep Product | awk '{print ($4,$5)}'
```

If on a Blade-based host, the slot number of the Blade in the chassis may be available through the command:

```
dmidecode -t 204 | grep 'Server Bay' | awk '{print $3}'
```

IBM Hosts: Model and Blade Slot

When the underlying hardware is IBM, start with the command:

```
dmidecode -t 1 | grep Product | awk '{print $4}'
```

The output can be several different strings, including **BladeCenter**, **System** and **xSeries** – a null string is also possible. In this last case, which seems to be applicable to pre-Lenovo IBM laptops, the command:

```
dmidecode -t 1 | grep Product | awk '{print $3}'
```

will return the IBM “Type” in the format **###???**.

If the first command returns a string containing **BladeCenter**, that indicates a Blade-based host, such as a something in an HS-20 BladeCenter. You can get the precise model number using:

```
dmidecode -t 1 | grep Product | awk '{print $6}'
```

Also, the slot number occupied by the Blade may be accessible using:

```
dmidecode -t 2 | grep 'Location In Chassis' | awk '{print substr($4,5,2)}'
```

If the result of the very first IBM-specific command was **System** or **xSeries**, then the underlying hardware is probably an xSeries rack-mounted system. You can obtain the model number with:

```
dmidecode -t 1 | grep Product | awk '{print $6}'
```

In the cases where the first command did not result in a null string, the subsequent commands will render the model number in the format **####???**; additionally, the string may contain extraneous characters on either or both sides of the model number itself. Further string processing will be needed to render the information in IBM's standard “Type” format of **####-???**.

Sun x86 Hosts: Model

This paper has had the benefit of few exemplars, so this information is limited. Start with the command:

```
dmidecode -t 1 | grep Product | awk '{print substr($4,1)}'
```

If the resulting string is **Fire**, then the underlying hardware is a Sun Fire model, and you can get the full model information using:

```
dmidecode -t 1 | grep Product | awk -F 'Fire' '{print $2}'
```

Otherwise, to glean more information, try using:

```
dmidecode -t 1 | grep Product | awk -F ': ' '{print $2}'
```

Physical and Virtual Host Serial Numbers

Compaq/HP and IBM Hardware

You should be able to extract the system's OEM-supplied serial number with the command:

```
dmidecode -t 1 | grep Serial | awk '{print $3}'
```

Note that it is possible for the result to be a null string. Typically, that can occur when the system's motherboard has been replaced and a serial number not assigned (a step that usually requires an OEM engineer to accomplish).

Sun Hardware

For reasons which aren't clear, **dmidecode** returns the motherboard serial number, not the serial number which appears on the chassis or anywhere else. To get the serial number as it appears in the ILOM and on the outside of the physical system box, you must use the IPMI toolset (which is available for, if not part of the default install on, most Linux distributions). If IPMI is running, the serial number should be retrievable using:

```
ipmitool fru | grep -A 6 'SYS (ID 3)' | grep 'Product Serial' | awk -F ': ' '{print $2}'
```

Virtualized Platforms

Virtualization platforms, such as VMware and KVM, do not offer Guest systems a “serial number” in the same manner of physical hardware OEMs. However, virtualized systems do have UUIDs, which can be used to much the same purpose.

To extract the UUID value on a virtual platform, use:

```
dmidecode -t 1 | grep UUID | awk '{print $2}'
```

Many physical platforms also have a UUID; whether or not they do, and what information they contain, varies from OEM to OEM. For the purposes of this paper, those are ignored in favor of the OEM-supplied Serial Number[4].

Ethernet Network Interfaces

Most Linux distributions offer four tools that are useful for probing Ethernet NICs: **ethtool** (which replaced **mii-tool**), **ifconfig**, **lspci** and **dmidecode**. These are each examined below with an eye to what information may be gleaned from them. For simplicity, the paper's examples will concentrate on interface **eth0**, but the techniques should work for most interfaces[5].

ethtool

The replacement for **mii-tool** offers a wealth of information about interfaces. Strangely, however, there doesn't seem to be any way to reliably get the MAC address for a NIC directly from this tool[6].

To determine the link state of an interface (which will be reported as either *yes* or *no*), use:

```
ethtool eth0 | grep Port | awk '{print $3}'
```

Probing the parameters of an interface that doesn't have link can produce very odd

results, so unless you're interested in something specific, determining link state is a good place to start.

For an “active” interface (one that is administratively up – see **ifconfig** below), you can determine the current link speed, duplex and transceiver type with the following commands:

```
ethtool eth0 | grep Speed | awk '{print $2}'
ethtool eth0 | grep Duplex | awk '{print $2}'
ethtool eth0 | grep Port | awk '{print $2}'
```

There some important caveats, particularly when dealing with Blade-based hosts. Depending on the Blade, an interface may report as having link, even when it technically does doesn't go anywhere[7]. Also, IBM Blade-based hosts may report a transceiver type of “FIBRE”, even when the actual media is copper[8]. Additional parsing of the **`ethtool eth0`** output can also (when supported) extract information concerning auto-negotiation state and capabilities, supported transceiver types, and supported link speeds and duplex combinations.

You can usually discover the PCI device number for an interface using:

```
ethtool -i eth | grep bus-info | awk '{print $2}'
```

Finally, VMware-based hosts using the *vmnics* and *pcnet32* driver combination will not report any of these information items.

ifconfig

This tool is primarily meant to examine and configure interfaces for OS usage. You can use it to count the number of Ethernet interfaces that the OS “knows about”:

```
ifconfig -a | grep -c eth
```

A positive integer should be the result, and it should also match the number of lines in /**etc**/**modprobe.conf** that begin with the string **alias eth**. For each interface in the system, one bit of pertinent hardware information to be found is usually the MAC address of the interface. For example, to get the MAC address of interface **eth0**, try using:

```
ifconfig eth0 | grep eth0 | awk '{print $5}'
```

You can also determine if the interface is *administratively up* (since it is possible for the OS to consider the interface “up” regardless of link state) using:

```
ifconfig | grep -A 1 eth0 | grep -c UP
```

The return will be **1** if the interface is administratively up.

lspci

Similar to the tool **lsscsi**, you can use **lspci** to examine the PCI devices discovered by the system during boot. With respect to NICs, the simplest thing is to count the number of Ethernet interfaces in the system:

```
lspci | grep -c Ethernet
```

The result (a positive integer) should be equal to the number of interfaces counted by the **ifconfig** command above. If it isn't, then a driver may be failing to load (for any number of reasons), or your system doesn't actually have all the interfaces listed in /**etc**/**modprobe.conf**, or your OS release may not have the necessary drivers.

You can get the PCI device number for each NIC as well:

```
lspci | grep Ethernet | awk '{print $1}'
```

As noted above, you can (in most instances) correlate the PCI device number to the interface name in the OS using **ethtool -i**. Use caution, however; the PCI device number may not be reported in the same format by both tools, and additional parsing may be required to correctly match one to the other.

By adding the **-v** parameter to **lspci**, you can get additional information for each Ethernet device; adding **-vv** will sometimes reveal even more information. Exactly what information can be gleaned, and how it should be parsed, depends greatly on the device in question. In a few cases, it may be possible to extract the interface's MAC address using **lspci**, although that seems to be the exception rather than the rule. The reader is encouraged to explore the possibilities on their own hardware.

dmidecode

Exactly what **dmidecode** can tell you about the network interfaces, and how you use it, is primarily determined by the underlying platform; generally, it is only useful for physical platforms, not virtual. In some cases, Blade *vs.* Rackmount may also make a difference.

IBM Hosts

On both IBM Blades and Rackmounts, start with:

```
dmidecode -t 8
```

This will list certain devices, differentiated by *Port Connector*. Many devices may be listed, including USB ports, SCSI HBAs and network interfaces. In terms of NICs, the output will look something like this example:

```
Handle 0x000E, DMI type 8, 9 bytes
Port Connector Information
      Internal Reference Designator: Not Specified
      Internal Connector Type: None
      External Reference Designator: Gigabit Ethernet
      External Connector Type: RJ-45
      Port Type: Network Port
```

The differentiator here is the final line. The string **Port Type: Network Port** typically indicates a network interface card. The string beginning **Handle 0x** gives a hexadecimal value, and it starts at various numbers (that is, the first value is not necessarily 0x0000), but will be presented in numerical order, and will have a definite range (*e.g.* from 000E to 001C). In a script, you'd have to determine the decimal equivalent of the first **Handle 0x** entry in the command output, and also the number of **Handle 0x** entries. Then you could construct a loop, using **grep** to extract each unique string starting with **Handle 0x**, and use the **-A** parameter to grab the 6 trailing lines of text so you can evaluate each block of output individually[9].

If, in an individual output block, you find the string **Network Port** on the **Port Type** line, then you probably have a NIC, and can parse the other lines for more information. In the example above, the **External Reference Designator** line indicates both that it is an Ethernet NIC and that it supports media speeds up to 1000Mb/s.

CAUTION

This command only reports embedded NICs, not add-on (in a bus slot) NICs.

A number of caveats apply to this technique. For example, Rackmounted systems may report an RSA interface as having the same **Port Type**, as shown in this example:

```
Handle 0x00DA, DMI type 8, 9 bytes
Port Connector Information
      Internal Reference Designator: Not Specified
      Internal Connector Type: None
      External Reference Designator: RSA Ethernet
      External Connector Type: RJ-45
      Port Type: Network Port
```

In that case, additional analysis of the **External Reference Designator** line would be needed to differentiate that interface from a NIC that the OS should see.

On IBM Blades (but not on Rackmounts), you can use the additional command:

```
dmidecode -t 10
```

The output will list certain items of *On Board Device* information; again, it will list more than just network interfaces. The output for a typical network interface would look like this:

```
On Board Device 2 Information
      Type: Ethernet
      Status: Enabled
      Description: Ethernet 1 BroadCom 5704S Ethernet Controller
```

Each Ethernet interface will have a **Description** string that begins with **Ethernet** and is followed by a number. The numbers for the interfaces will start at 1 and go up in numerical order, making it easy to program into a script loop. The line **Type: Ethernet** is the determinant for a network interface. The **Status** line indicates if the NIC has been disabled in BIOS, and the full **Description** text closely (but not exactly) matches the descriptive text retrieved by **lspci**.

HP/Compaq

The **-t 8** and **-t 10** command-line options are not useful on HP hardware. Instead, use the command:

```
dmidecode -t 209
```

This reports similar information in a single block limited to just NICs (so no parsing multiple entries trying to distinguish NICs from other devices; the *Handle* may also be ignored), as shown in this example:

```
Handle 0xD100, DMI type 209, 36 bytes
HP BIOS NIC PCI and MAC Information
      NIC 1: PCI device 02:02.0, MAC address 00:15:60:AA:84:00
      NIC 2: PCI device 02:02.1, MAC address 00:15:60:AC:74:FF
      NIC 3: PCI device 05:01.0, MAC address 00:16:35:C5:CD:B0
      NIC 4: PCI device 05:02.0, MAC address 00:16:35:C6:CD:AF
```

From this output, the PCI device number and MAC address can be extracted for each on-board NIC. Media, connector type and other descriptive information will have to come from elsewhere.

CAUTION

This command only reports embedded NICs, not add-on (in a bus slot) NICs. Also, while **NIC 1** will usually refer to **eth0**, there is no guarantee of such a correlation. Finally, iLO NICs are not listed.

Sun

Similar to IBM hardware, start with the command:

```
dmidecode -t 8
```

While the output is very similar to that on IBM hosts, the analysis is not:

```
Handle 0x0024, DMI type 8, 9 bytes
Port Connector Information
      Internal Reference Designator: J13
      Internal Connector Type: None
      External Reference Designator: LAN0
      External Connector Type: RJ-45
      Port Type: Network Port
```

Among other differences, it is important to note that the **External Reference Designator** for NICs is in the form **LAN#**. However, it seems that Sun ILOM interfaces are not detected by this command, so that makes analysis somewhat easier.

Also, similar to IBM Blades, you can get some additional information with:

```
dmidecode -t 10
```

This will list certain items of *On Board Device* information; again, it will list more than just network interfaces. The output for a typical network interface would look like this:

```
Handle 0x0041, DMI type 10, 6 bytes
On Board Device Information
      Type: Ethernet
      Status: Enabled
      Description:  Gigabit Ethernet #1
```

CAUTION

Unlike the output from IBM Blades, the *On Board Device* lines do not have sequential numbers differentiating each information block. Instead, the **Handle 0x** line must be used. The **Type** and **Status** lines appear to use the same values as the IBM Blades, but the **Description** line has an additional numeric identifier (starts counting at 1, but digits may also appear in other **Description** lines not associated to NICs).

KVM and VMware

The **dmidecode** tool is not useful for extracting NIC information on these platforms.

RAM

On both physical and virtual platforms, you can retrieve information about the RAM installed on the host with the command:

```
dmidecode -t 17
```

The returned output is similar to:

```
Handle 0x001E, DMI type 17, 27 bytes
Memory Device
      Array Handle: 0x001D
      Error Information Handle: No Error
      Total Width: 32 bits
      Data Width: 32 bits
      Size: 2048 MB
      Form Factor: DIMM
      Set: None
      Locator: RAM slot #0
      Bank Locator: RAM slot #0
      Type: DRAM
      Type Detail: EDO
      Speed: Unknown
      Manufacturer: Not Specified
      Serial Number: Not Specified
      Asset Tag: Not Specified
      Part Number: Not Specified

Handle 0x001F, DMI type 17, 27 bytes
Memory Device
      Array Handle: 0x001D
      Error Information Handle: No Error
      Total Width: 32 bits
      Data Width: 32 bits
      Size: 2048 MB
      Form Factor: DIMM
      Set: None
      Locator: RAM slot #1
      Bank Locator: RAM slot #1
      Type: DRAM
      Type Detail: EDO
      Speed: Unknown
      Manufacturer: Not Specified
      Serial Number: Not Specified
      Asset Tag: Not Specified
      Part Number: Not Specified
```

For analysis, you can use **grep** to collect all the **Size** lines, then split out the numeric value (which should always be in MB) and compute the total. Here is a sample script:

```
#!/bin/bash
TOTALRAM=0
RAMSTICKS=`dmidecode -t 17 | grep 'MB' | grep 'Size:' | awk '{print $2}'`
for RAMSTICK in ${RAMSTICKS[@]}; do
      (( TOTALRAM = TOTALRAM + RAMSTICK ))
done
echo "The system has ${TOTALRAM} MB of RAM installed"
```

A KVM host will report total RAM as a single memory DIMM in a single slot. Hosts in a VMware environment may not report the same RAM as is allocated to the VM in the VIC.

CPU Analysis

While analyzing CPUs might seem relatively simple, that can be deceiving. Some basic information is straightforward. For example, the command:

```
grep -m 1 'vendor_id' /proc/cpuinfo | awk '{print $3}'
```

will return a string – within the scope of this paper, the two most-likely values are **GenuineIntel** or **AuthenticAMD**, although simply **Intel** or **AMD** may also be returned.

When dealing with Intel CPUs, you should be able to get the CPU's Model with the command:

```
grep -m 1 'model name' /proc/cpuinfo | awk '{print $5}' | awk -F '(' '{print $1}'
```

AMD is not quite as straightforward; start with the command:

```
grep -m 1 'model name' /proc/cpuinfo | awk '{print $8}'
```

If that returns a null string, then try:

```
grep -m 1 'model name' /proc/cpuinfo | awk '{print $7}'
```

In both cases, the string will be a generic name used for marketing (*e.g.* Xeon for Intel, 8218 for AMD). If you want to get more precise, then examine the lines for *cpu family*, *model* and *stepping*.

You can also examine the *flags* line if you need to determine whether or not a particular CPU feature is supported. For example, if the *flags* line contains the string **lm**, then the CPU supports 64-bit long mode (note that if your kernel is a 32-bit kernel, the flag may not be present even if the CPU does have the support). The string **ht** or **htt** indicates *HyperThreading* support in the CPU (although it may have been disabled in BIOS).

Individual physical CPUs can be distinguished by each unique value in the *physical id* field, although any given value will appear more than once when dealing with a multi-core (or HyperThreaded) CPU type. The *cpu cores* field will show the number of cores per CPU, but won't help distinguish a HyperThreaded CPU from an actual unicore CPU.

You can also gather much the same information in a slightly more-processed format with the command:

```
dmidecode -t 4
```

The outputs are very similar, although referencing **/proc/cpuinfo** provides you with somewhat more elemental information. The route you should choose depends on your needs and situation.

Fiber HBAs

The statements loading the HBA drivers will be in **/etc/modprobe.conf** and appear on lines containing the string **scsi_hostadapter**. Drivers for other SCSI devices (*e.g.* RAID arrays, CD-ROM drivers, *etc.*) also appear on similar lines, so it is necessary to examine the driver name on the same line to determine if it is an HBA driver. You can also use **lsmod** to see if the specific kernel modules for your HBA brand/model are loaded.

This paper will focus on detecting and analyzing two of the more common HBA manufacturers, Emulex and Qlogic. The techniques may or may not transfer well to other OEMs.

Depending on the kernel and driver versions, information about HBAs can be found under either **/sys/class/scsi_host** or **/proc/scsi** (the latter location is typical for older kernel and driver versions). Beyond detecting which drivers are loaded, analyzing the HBA information requires parsing information in files under those directories.

Emulex

For each HBA, there should be a sub-directory in **/sys/class/scsi_host** and the sub-directory will be named **host#** where **#** is a digit. Note that other SCSI devices (such as RAID adapters) will have entries in the same place, and following the same naming convention. You can distinguish a sub-directory associated with an Emulex HBA by looking for a file named **modeldesc** - if it is present and contains the string **Emulex**, then the sub-directory is associated with an Emulex HBA.

In that sub-directory, the **modelname** file will contain the Model of the HBA. The **lpfc_drvr_version** file contains the version of the **lpfc** driver; the **fwrev** file has the version of

the HBA's firmware. The WWPN is in the **node_name** file and the adapter port's state can be determined by examining the **state** file. The media speed can be found in the **speed** file.

QLogic

In older kernel/driver environments, start by looking for subdirectories of /**proc**/**scsi** with names that start with the letters **qla** (possible names include *qla2xxx* and *qla2300*). When present, information can be extracted by examining the single file (per HBA) in the directory structure. The file name will be a number (*e.g.* 1); while the numbers will be sequential, counting does not necessarily begin at 0. Typically, the HBA's Model, Firmware Version, WWPN and Port State can be determined by parsing the file associated with the HBA. The version of the driver should also be accessible.

For environments with newer kernel/driver versions, for each HBA there will be a sub-directory in /**sys**/**class**/**scsi_host**. The sub-directory will be named **host#** where **#** is a digit. Note that other SCSI devices (such as RAID adapters) may have associated directories in the same place, and following the same naming convention. You can distinguish a sub-directory associated with a QLogic HBA by looking for a file named **model_name** - if it is present, then the sub-directory is associated with a QLogic HBA and the contents of the file will have the Model of the HBA.

Also in that sub-directory, the **driver_version** file contains the version of the QLogic driver; the **fw_version** file has the version of the HBA's firmware. The WWPN is in the **port_name** file and the adapter port's state can be determined by examining the **state** file.

Tips for Probing Hardware

First, decide what information you need to find. Is it enough to determine that you're running on an AMD CPU, or is it important to distinguish an 885 from an 8218? For your purposes are all Ethernet interfaces the same, or is separating BroadCom from RealTek crucial?

Once you have a clear idea of the depth to which you'll need to plumb, look at your tools. Is the Swiss Army knife called **dmidecode** sufficient? Does it reliably offer all the information you need, or will you have to dig?

If you're going to have to dig, it simply becomes a matter of figuring out where the OS, or the driver, squirrels away the data you want. Looking under /**sys**/**class** or /**proc** are usually the best places to start. Use **lsof** to determine what files the driver opens, and check **dmesg** for additional hints and pointers[10].

Footnotes

[1] Sometimes one can find devices *via* a generic driver, such as some PCI devices, even if the OS did not load the correct driver or kernel modules for the specific bit of hardware. Doing so is outside the scope of this paper.
[2] There are, of course, any number of other approaches, such as (to mention one example) looking for /**proc**/**vmcore** (indicating a KVM system, either a Guest or a Host) or looking for specific kernel modules (*e.g.* **kvm** or **vmmemctl**) listed in /**proc**/**modules**. This paper is presenting a generic starting point.
[3] http://www.demidecode.org
[4] For the purposes of this paper, the primary use for a Serial Number is to differentiate one system on a particular hardware type from another system on the same hardware type. The combination of the OEM, Model and Serial Number can be thought of as a unique key for identifying the hardware.
[5] Some virtualized network interfaces are problematic.
[6] In some cases, it is possible to retrieve the Burned-In Address using **ethtool -e**. However, even when the information is present, the offset (within the output stream) is highly variable (depending on the NIC, the driver and whether or not the NIC is part of a logical bond). Readers are encouraged to experiment with their own hardware.
[7] This can occur when the blade chassis switch port underlying the blade's NIC is enabled, but it not actually connected to anything (or is not configured to properly transport the blade's IP traffic).
[8] The fact that this can happen indicates just how complex the underlying hardware architecture can be.
[9] Naturally, there are many other methods and tools to parse and analyze the output. Choose those applicable to your environment and goals.
[10] Occasionally, you can dig bits of information out of **dmesg** that can't be found anywhere else, if for no other reason than the bit of information is only present at the time a driver initializes.

Change Log

Version	Date	Change
0.10	2010-Aug-10	Initial creation
0.20	2010-Sep-28	Revised and expanded
0.30	2010-Oct-25	Revised and expanded
0.40	2010-Nov-11	Revised and expanded
0.50	2010-Dec-13	Revised for publication
1.00	2011-Jan-14	Published

End of Document