VMware Backdoor I/O Port

NOTICE

Now you can get more accurate information by studying the Open Virtual Machine Tools source code.
This page predates the Open Virtual Machine Tools and all information here came from my own
research and contribution by many VMware users around the world.

What is "Backdoor"?

This text describes how VMware Command Line Tools communicates with VMware running on the host machine. Apparently the official VMware-Tools uses the same method.

Basic idea is that the communication is done by accessing a special I/O port specific to VMware virtual machines (The "Backdoor" port in this text). The following operation invokes Backdoor functions:

```
/* in Intel syntax (MASM and most Windows based assemble

MOV EAX, 564D5868h                    /* magic number
MOV EBX, command-specific-parameter
MOV CX,  backdoor-command-number
MOV DX,  5658h                        /* VMware I/O Po

IN  EAX, DX (or OUT DX, EAX)

/* in AT&T syntax (gnu as and many unix based assemblers

movl $0x564D5868, %eax;               /* magic number
movl command-specific-parameter, %ebx;
movw backdoor-command-number, %cx;
movw $0x5658, %dx;                    /* VMware I/O po

inl  %dx, %eax; (or outl %eax, %dx)
```

In appearance it is just a straight forward I/O access operation. In actuality the *IN* (or *OUT*) instruction here plays a quite different role from usual

circumstances - it is rather a trigger of the Backdoor function than an I/O access instruction.

The example shows that the magic number and other values are stored in *EAX*,*EBX* and *CX* registers **prior to** the *IN/OUT* instruction, although normally *IN* instruction is affected only by *DX* and *OUT* by *EAX/DX*, because the Backdoor function uses them as input (and/or output) parameters. In fact the Backdoor function may use **any** registers to pass values in both direction. Though the use of only *EAX*, *EBX*, *CX* and *DX* is shown in the example, certain Backdoor commands also use high words of *ECX* and *EDX* as well as *ESI*, *EDI* and *EBP* both for input parameters and to store results on return.

Only difference between *IN* and *OUT* instructions is that *OUT* instruction does not return a value in *EAX*. Otherwise both instructions have exactly the same effect.

Consequently, it is very unlikely that you can use high level language library functions to access this Backdoor port (e.g. _inp()/_outp() functions available in some C runtime libraries), because they never expect these registers to be modified by *IN/OUT* instructions.

TOP

Backdoor commands

The following information is gathered mostly on VMware products for Windows host. I'd appreciate it if you let me know if you found errors, differences in Linux hosts, or information on products not listed here.

Command Number	Description	Availability Y: available (Y): available but function unknown N: not available ?: availability unknown WS4.0 and 4.5 are listed separately because of their significant difference						
		Workstation					GSX	
		2.x	3.x	4.0	4.5	5.x	2.5	3.2
00h	? (likely to be unused)	N	N	N	N	N	N	N
01h	Get processor speed (MHz)	Y	Y	Y	Y	Y	Y	Y
02h	APM function	Y	Y	Y	Y *1	Y *1	Y	Y *1
03h	? (likely keyword: getDiskGeometry)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)
04h	Get mouse cursor position	Y	Y	Y	Y	Y	Y	Y

05h	Set mouse cursor position	Y	Y	Y	Y	Y	Y	Y
06h	Get text length from clipboard	Y	Y	Y	Y	Y	Y	Y
07h	Get text from clipboard	Y	Y	Y	Y	Y	Y	Y
08h	Set text length to clipboard	Y	Y	Y	Y	Y	Y	Y
09h	Set text to clipboard	Y	Y	Y	Y	Y	Y	Y
0Ah	Get VMware version	Y	Y	Y	Y	Y	Y	Y
0Bh	Get device information	Y	Y	Y	Y	Y	Y	Y
0Ch	Connect / disconnect a device	Y	Y	Y	Y	Y	Y	Y
0Dh	Get GUI option settings	Y	Y	Y	Y	Y	Y	Y
0Eh	Set GUI option settings	Y	Y	Y	Y	Y	Y	Y
0Fh	Get host screen size	Y	Y	Y	Y	Y	Y	Y
10h	? (likely keyword: monitorControl)	(Y)	N	N	N	N	N	N
11h	Get virtual hardware version	(Y)*2	N	N	N	Y	N	N
12h	Popup "OS not found" dialog	N	Y	N*3	Y *1	Y *1	Y	Y *1
13h	Get BIOS UUID	N	N	Y	Y	Y	Y	Y
14h	Get memory size (MB)	N	Y	Y	Y	Y	Y	Y
15h	?	(Y)	N	N	N	N	N	N
16h	? (likely keyword: getOS2IntCursor)	(Y)	(Y)	(Y)	(Y)	N	(Y)	(Y)
17h	Get host's system time (GMT)	Y	Y	Y	Y	Y	Y	Y
18h	? (likely keyword: stopCatchup)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)
19h	? (likely to be unused)	N	N	N	N	N	N	N
1Ah	? (likely to be unused)	N	N	N	N	N	N	N
1Bh	? (likely to be unused)	N	N	N	N	N	N	N
1Ch	? (likely keyword: initScsiOprom)	N	N	N	(Y)	(Y)	N	(Y)
1Dh	? (likely to be unused)	N	N	N	N	N	N	N

1Eh	Guest to host RPC	Y	Y	Y	Y	Y	Y	Y
	Enhanced RPC	N	N	Y	Y	Y	N	Y
1Fh	? (likely keyword: rsvd0)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)
20h	? (likely keyword: rsvd1)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)
21h	? (likely keyword: rsvd2)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)
22h	? (likely keyword: ACPID)	N	N	(Y)	(Y)	(Y)	N	(Y)
23h	? (likely to be unused)	N	N	N	N	N	(Y)	N
24h	? (likely to be unused)	N	N	N	N	N	N	N
25h	? (likely keyword: PatchSMBIOS)	N	N	N	(Y)	(Y)	N	(Y)
26h	? (likely to be unused)	N	N	N	N	N	N	N
27h	? (likely keyword: absMouseData)	N	N	N	(Y)	(Y)	N	(Y)
28h	? (likely keyword: absMouseStatus)	N	N	N	(Y)	(Y)	N	(Y)
29h	? (likely keyword: absMouseCommand)	N	N	N	(Y)	(Y)	N	(Y)
2ah	? (likely to be unused)	N	N	N	N	N	N	N
2bh	? (likely keyword: PatchACPITables)	N	N	N	N	(Y)	N	N

*1: Work only if the virtual processor is in privileged mode (e.g. boot BIOS, real mode DOS).
*2: Seems to be implemented as an entirely different function.
*3: Implemented but works only in the context of virtual machine BIOS, practically unavailable from guest OS.

TOP

01h - Get processor speed (MHz)

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0001h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = Processor speed in MHz
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command returns the host machine's processor speed. Note that the returned value is a value estimated (calculated) by VMware program. For example, I usually get 3EAh (1,002) on my 1000MHz machine.

This information is originally reported by Andrei Tarassov.

TOP

02h - APM function

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5(*) WS5.x(*) GSX2.5 GSX3.2(*)

CALL
EAX = 564D5868h - magic number
EBX = APM function specific parameter
ECX(HI) = APM function number (see below)
ECX(LO) = 0002h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = ? / FFFFFFFFh: failure
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command invokes a certain APM function of the virtual machine. Parameter combinations currently known to work are:

APM func (ECX:HI)	APM param (EBX:LO)	note
0007h	0004h	suspend. **DO NOT USE**- Though this command causes the virtualmachine to suspend, it seems not to be fully compatible with VMware suspend feature and the virtual machines cannot be resumed
0007h	0007h	power off

(*) On WS4.5/GSX3.2 and later, this command works only if the virtual processor is in privileged mode (such as in real mode DOS and in OS kernel), apparently for security reasons (I guess I'm partly to blame for this, by revealing

the backdoor information...). Works without such restrictions on WS4.0/GSX2.5 and earlier.

TOP

04h - Get mouse cursor position

AVAILABILITY

WS2.x WS3.x WS4.0 WS4.5(*) WS5.x(*) GSX2.5 GSX3.2(*)

CALL

EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0004h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN

EAX(HI) = X coordinate
EAX(LO) = Y coordinate
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION

This command returns the mouse cursor position relative to the upper-left corner of the guest screen area.
It returns FF9CFF9Ch (-100, -100) when the guest does not have the focus so this command can be used to detect if the guest has the focus.

(*) On WS4.5/GSX3.2 and later, when VMware preference option "Ungrab when cursor leaves window" is enabled, VMware keeps track of mouse movement even if the mouse cursor is not shown in the guest (such as in DOS, linux console), and this command causes VMware to release the focus from the guest if the supposed cursor position at that moment is outside the guest screen area.
With this option disabled VMware does not keep track of the cursor position and this command always returns the last known cursor position (the position at the time when the guest grabbed the input focus, or the position set with command 05h) as earlier versions do.

TOP

05h - Set mouse cursor position

AVAILABILITY

WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX(HI) = X coordinate
EBX(LO) = Y coordinate
ECX(HI) = don't care
ECX(LO) = 0005h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command changes the mouse cursor position relative to the guest screen area. For example, if you set cursor position to (0, 0), mouse cursor appears in the host at the upper-left corner of the guest screen area when you release cursor from the guest (unless "Ungrab when cursor leaves window" VMware preference option is enabled on WS4.5/GSX3.2 and later - see description of command 04h).

The cursor position does not have to be inside the guest screen area, although coordinate values are interpreted as unsigned values so you can not specify negative values (if you specify (-1, -1), it is interpreted as (65535, 65535) and you'd get the cursor beyond the right-bottom edge of the host). Even if you move the cursor outside the guest screen area, the cursor does not appear in the host until the guest loses the focus.

TOP

06h - Get text length from clipboard

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5(*) WS5.x(*) GSX2.5 GSX3.2(*)

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0006h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = text length
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION

This command returns the length of text data available from the host's clipboard. Also this command resets the clipboard data transfer process. The first get text command (07h) issued after this command returns the very beginning of the text data.

On WS4.0/GSX2.5 and earlier, the maximum possible length is 65335. When longer text is stored in the clipboard, 0 is returned.
On WS4.5/GSX3.2 and later, the maximum possible length is 65436. When longer text is stored in the clipboard, this command still returns 65436 and data is clipped at this length. FFFFFFFFh is returned if the data in the clipboard is not a text or the data is already transfered once (see notes below).

On WS4.0/GSX2.5 and earlier, this command often returns slightly larger value (by 5 or so) than the actual length of data available. Also on Windows host each CR/LF combination (0Dh+0Ah) is counted as 2 characters, although subsequent get data command (07h) does not pass 0Dh characters at all.
WS4.5/GSX3.2 and later returns the exact length of data available with get data command (not including the terminating NULL character).
For example, if you have "xyz[0Dh][0Ah]" in the clipboard, WS4.0/GSX2.5 and earlier is likely to return 10, but you'll get only 4 characters "xyz[0Ah]" in the guest. WS4.5/GSX3.2 and later returns exactly 4 in such case.

(*)Notes on WS4.5/GSX3.2 and later

On WS4.5/GSX3.2 and later, pasting operations (combination of 06h and 07h) work only once after the virtual machine gets the focus. You can reset the pasting process with command 06h and start over as many times as you like until the last byte of the data is transferred. However after the last byte of the data is delivered, get length command (06h) always returns FFFFFFFFh and get text command (07h) always returns 00000000h, and you cannot get the same data again, unless the focus is released and grabbed again.
Also copying operations (combination of 08h and 09h) work only after you have completed a pasting operation to the last byte of the data. If you execute copying operations without completing a pasting operation, VMware simply ignores the copying commands. Once you have completed a pasting operation, you can perform copying operations repeatedly.
As a consequence of these behaviors, you cannot paste the data you copied into the clipboard with copying operations, unless the focus is released/re-grabbed between the two operations.

```
(grab)->paste->paste = NG
(grab)->copy = NG
(grab)->incomplete paste->copy = NG
(grab)->paste->copy->copy = OK
(grab)->paste->copy->paste = NG
(grab)->paste->copy->(release/grab)->paste = OK
```

This behavior is first reported by Yoshiki Shibukawa.

TOP

07h - Get text from clipboard

AVAILABILITY

WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL

EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0007h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN

EAX = 4 bytes of text from clipboard (first byte in LSB)
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION

This command return a portion of text in the clipboard. The get text length command (06h) should be called prior to this command. The first call after a command 06h call returns the first 4 bytes, and the next call returns the next 4 bytes, and so on.
If no more data is available in the clipboard, 00000000h is returned.
This command does not pass carrage return characters (0Dh), so if the guest OS requires them (e.g. DOS), you have to supply them as line feed characters (0Ah) appear in returned text data.

On WS4.0/GSX2.5 and earlier the data length value returned by the get length command (06h) is often slightly larger than actual text length so you should search for a terminating null character in returned text data to know the actual end of the text.

See notes on WS4.5/GSX3.2 and later in command 06h above.

TOP

08h - Set text length to clipboard

AVAILABILITY

WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL

EAX = 564D5868h - magic number
EBX = text length (not including a terminating NULL)
ECX(HI) = don't care
ECX(LO) = 0008h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
Use this command before actually sending text data to the clipboard, or to reset the transfer process and start over.
The length does not have to include the terminating NULL character.

See notes on WS4.5/GSX3.2 and later in command 06h above.

TOP

09h - Set text to clipboard

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX = 4 bytes of text to set (first byte in LSB)
ECX(HI) = don't care
ECX(LO) = 0009h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command send text data to the clipboard, in the same manner as the get text command (07h).

On windows host, VMware seems to unconditionally supply a carriage return character (0Dh) for each line feed character (0Ah) transfered by this command, so you should not pass carriage return characters to this command (if you pass a '0Dh 0Ah' sequence, you'll get a '0Dh 0Dh 0Ah' in the host.)

See notes on WS4.5/GSX3.2 and later in command 06h above.

TOP

0Ah - Get VMware version

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX = don't care (any value different from magic number)
ECX(HI) = don't care
ECX(LO) = 000Ah - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = version number (?)
EBX = 564D5868h - magic number
ECX = product type on WS3.x/GSX2.5 and later (see below) / unchanged on WS2.x
EDX = unchanged

DESCRIPTION
You can use this command to decide if it is running inside VMware. If the backdoor magic number is returned to *EBX*, then you can be certain that it is running inside VMware.

On WS3.x/GSX2.5 and later, the value returned in *ECX* indicates the VMware product type. The following values are currently known:

- 01h = Express
- 02h = ESX Server
- 03h = GSX Server
- 04h = Workstation

I have no idea what the value returned in *EAX* means. For the record, it is always 6 as far as I know (VMware WS2,3,4,5 and GSX2,3, all on Windows host).

TOP

0Bh - Get device information

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX(HI) = device number (0 to 51)
EBX(LO) = data offset (0 to 36)
ECX(HI) = don't care
ECX(LO) = 000Bh - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = 000000001h: success / 00000000h: failure
EBX = device information data (4 bytes at the specified offset, first byte in LSB)

ECX = unchanged
EDX = unchanged

DESCRIPTION
This command returns a portion of device information data for connectable virtual devices. Device information data for each device consists of 40 bytes and you have to call this command 10 times for each device with different offset values (0, 4, 8, ...) to get complete device information data.

Data offset value does not have to be aligned to 4 bytes boundary (you could specify any value between 0 and 36).

Device information data consists of:

- offset 0..?: device name (such as "floppy0", "ide1:0", and "sound")
- offset 36: 01h = connected / 00h = disconnected

Any other field is unknown yet.

This command fails if the specified device does not exist, or specified offset is larger than 36.

TOP

0Ch - Connect / disconnect a device

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x(*) GSX2.5 GSX3.2(*)

CALL
EAX = 564D5868h - magic number
EBX(HI) = 8000h (connect) / 0000h (disconnect)
EBX(LO) = device number (0 to 51)
ECX(HI) = don't care
ECX(LO) = 000Ch - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = 00000001h: success / 00000000h: failure
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command connects / disconnects a connectable virtual device.

(*)On WS5.5/GSX3.2 (and possibly earlier also) this command may fail if you have not issued the command 0Bh at least once after powering on the virtual machine.

TOP

0Dh - Get GUI option settings

AVAILABILITY

WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL

EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 000Dh - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN

EAX = option settings bitmask (see below)
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION

This command returns the current option settings as a bitmask value. Each bit corresponds to the following option item:

- 0001h: grab when cursor enters window
- 0002h: ungrab when cursor leaves window
- 0004h: scroll when cursor approaches window edge
- 0010h: copy and paste between host and guest
- 0040h: indicates if running in fullscreen - see the next command
- 0080h: switch to fullscreen - see the next command
- 0400h: time syncronization between host and guest

TOP

0Eh - Set GUI option settings

AVAILABILITY

WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL

EAX = 564D5868h - magic number
EBX = option settings bitmask
ECX(HI) = don't care
ECX(LO) = 000Eh - command number
ECX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN

EAX = ?
EBX = unchanged

ECX = unchanged
EDX = unchanged

DESCRIPTION
This command changes GUI option settings. The same bitmask as the command 0Dh is used.

Setting the bit 0040h (fullscrren indicator) with this command does not have an effect. Instead, you can set the bit 0080h to switch to fullscreen mode. On WS3.x/GSX2.5 and earlier, however, if you switch to fullscreen in this manner the original VMware window becomes invisible and you can no longer use VMware in windowed mode until you shutdown the guest and exit VMware application (on Windows host. Could be different on Linux host).

TOP

0Fh - Get host screen size

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX 3.2

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 000Fh - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX(HI) = X resolution (pixels)
EAX(LO) = Y resolution (pixels)
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command returns the host's screen size.

TOP

11h - Get virtual hardware version

AVAILABILITY
WS5.x

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care

ECX(LO) = 0011h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = virtual hardware version
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command returns the virtual hardware version of the current virtual machine.
Possible version numbers are:

- 3: Virtual machines created with WS4.x, ESX2.x, GSX3.x, ACE1.x, and with WS5.x as a legacy VM
- 4: Virtual machines created with WS5.x as a new type VM

Although virtual machines created with WS3.x/GSX2.x also have a virtual hardware version (1 or 2), they can not run on WS5.x without first upgrading the virtual hardware and therefore this command never returns such values.

Note: Command 11h is also implemented in WS2.x but it seems to have a different function and I don't know what.

TOP

12h - Popup "OS not found" dialog

AVAILABILITY
WS3.x WS4.5(*) WS5.x(*) GSX2.5 GSX3.2(*)

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0012h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = unchanged
EBX = unchanged
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command causes VMware application to display a "A bootable CD-ROM disc, floppy diskette, or hard disk was not detected." dialog box in the host (the

text may differ slightly by products and versions). The virtual machine is completely frozen until the dialog box is dismissed.

(*) On WS4.5/GSX3.2 and later this command works only when the virtual processor is in privileged mode, such as is real mode DOS and OS kernel. The command is actually implemented also in WS4.0 but is restricted to work only in the context of virtual machine BIOS and practically unavailable in the guest OS. On WS3.x/GSX2.5 it works without restriction.

TOP

13h - Get BIOS UUID

AVAILABILITY
WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0013h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = 1st 4 bytes of the UUID (first byte in LSB)
EBX = 2nd 4 bytes of the UUID (ditto)
ECX = 3rd 4 bytes of the UUID (ditto)
EDX = 4th 4 bytes of the UUID (ditto)

DESCRIPTION
This command returns the BIOS UUID of the current virtual machine. BIOS UUID is stored in the config file in the following form:

```
uuid.bios = "56 4d 3e 7a 92 ee 4c 46-e8 0d 86 f3 68 a
```

With this command, the UUID illustrated above is returned in each registor in the following form:

```
EAX: 7a3e4d56
EBX: 464cee92
ECX: f3860de8
EDX: e7cba068
```

TOP

14h - Get memory size (MB)

AVAILABILITY
WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0014h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = memory size in MB
EBX = ?
ECX = unchanged
EDX = unchanged

DESCRIPTION
This command returns the memory size assigned to the virtual machine. Seemingly meaningful value is also returned to *EBX*, but I counldn't figure out the meaning of that value yet (always 00001177h as far as I know).

TOP

17h - Get host's system time (GMT)

AVAILABILITY
WS2.x WS3.x WS4.0 WS4.5 WS5.x GSX2.5 GSX3.2

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = don't care
ECX(LO) = 0017h - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = host's system time (GMT, unix style 32 bit time value)
EBX = microsecond
ECX = ?
EDX = host's timezone (offset to GMT in minutes, 32 bit signed integer) on WS4.0/GSX2.5 and earlier / 0 on WS4.5/GSX3.2 and later

DESCRIPTION
This command returns the host's system time (GMT).
On WS4.0/GSX2.5 and earlier, you can get the host's local time by subtracting the offset (*EDX*) from the GMT time (*EAX*). Since the offset is returned in minutes rather than in seconds, the expression should be like:

```
localtime = EAX - (EDX * 60)
```

Another seemingly meaningful value is returned in *ECX* but I still don't know what it is. For the record, on my WS5.5 it is almost always 000F4240h

(1,000,000) but somethimes changes between 1,000,000 and about 3,000,000. I'd imagine it has something to do with the clock precision. Maybe.

TOP

1Eh - Guest to host RPC

AVAILABILITY

WS2.x WS3.x WS4.0(*) WS4.5(*) WS5.x(*) GSX2.5 GSX3.2(*)

CALL

EAX = 564D5868h - magic number
EBX = subcommand specific parameter
ECX(HI) = RPC subcommand
ECX(LO) = 001Eh - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN

EAX = ?
EBX = subcommand specific result
ECX = subcommand specific result
EDX = subcommand specific result

DESCRIPTION

This command is used to invoke a guest-to-host RPC command.
The following subcommands are used to invoke a single RPC command, usually in this order:

- 00h: open RPC channel
- 01h: send RPC command length
- 02h: send RPC command data
- 03h: receive RPC reply length
- 04h: receive RPC reply data
- 05h: finish receiving RPC reply
- 06h: close RPC channel

(*) WS4.0/GSX3.2 and later also supports a slightly different (and I suppose faster) data transfer mechanism which involves ESI, EDI and EBP registers and another I/O port (the old method still works).
This mechanism is described in the Enhanced RPC section.
The information on enhanced RPC was first reported by Nickolai Zeldovich.
Some additional information was supplied by Ryoji Kanai.

Following RPC commands are currently known:

- "machine.id.get"
  Retrieves the "machine.id" string specified in the virtual machine config file or vmware command line.
  The reply should be "1 <machine id string>" on success or "0 No

machine id&quot

- "log <message>"
  Outputs "message" into the vmware.log file in the host.
  The reply should be "1 " on success or "0 ".

- "info-set guestinfo.<varname&gt <value>"
  Associates the <value> to the <varname>.
  The reply should be "1 " on success or "0 ".

- "info-get guestinfo.<varname>"
  Retrieves the value assigned to <varname> with "info-set" RPC command.
  The reply should be "1 <value>" on success or "0 No value found".

- "tools.set.version <version>
  Reports the vmware-tools version to VMware. Apparently VMware uses this information to display "install/update your vmware-tools" message in the status area.
  The reply should be "1 " on success or "0 ".

- "disk.wiper.enable"
  Queries if disk shrinking is possible on the current virtual machine (i.e. the virtual machine has no snapshot).
  The reply should be "1 1" if shrinking is possible, "1 0" otherwise, or "0 " on other errors.

- "disk.shrink"
  Starts disk shrinking process. Note that cleaning unused spaces in the file system on virtual disks is the guest's responsibility (That's what official VMware-Tools is actually doing when it is "preparing" virtual disks for shrinking).
  The reply should be "1 " on success or "0 " on error (or cancelled by user).
  Shrinking disks when the virtual machine has a snapshot does not cause an error. The process finishes immediately but is reported as success.

- "f <binary data>"
  Used by Shared Folder and Drag & Drop features to exchange file information/data between the host and the guest. Details unrevealed yet.

TOP

RPC subcommand 00h - Open RPC channel

CALL

EAX = 564D5868h - magic number

EBX = 49435052h - RPC open magic number ('RPCI')
ECX(HI) = 0000h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = 00010000h: success / 00000000h: failure
EDX(HI) = RPC channel number
EDX(LO) = 0000h

DESCRIPTION
Opens an RPC channel. Subsequent RPC subcommands use the channel number returuned in the high order word of EDX.

Up to 8 channels (#0 - #7) can be opened simultaniously on a single virtual machine. When you try to open more than that, VMware simply starts over from #0 and does not report any error.

TOP

RPC subcommand 01h - Send RPC command length

CALL
EAX = 564D5868h - magic number
EBX = command length (not including the terminating NULL)
ECX(HI) = 0001h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = 00810000h: success / 00000000h: failure
EDX = unchanged

DESCRIPTION
Send RPC command length. An RPC channel must be open before invoking this command.

TOP

RPC subcommand 02h - Send RPC command data

CALL
EAX = 564D5868h - magic number
EBX = 4 bytes from the command data (the first byte in LSB)

ECX(HI) = 0002h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = 000010000h: success / 00000000h: failure
EDX = unchanged

DESCRIPTION
Send RPC command data by 4 bytes at a time. The RPC command length must be sent with subcommand 01h prior to this command. For example, to send the RPC command "machine.id.get" you have to call this subcommand 4 times with EBX set to 6863616Dh ("mach"), 2E656E69h ("ine."), 672E6469h ("id.g") and 00007465h ("et\x00\x00") in this order.

TOP

RPC subcommand 03h - Recieve RPC reply length

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = 0003h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = reply length (not including the terminating NULL)
ECX = 00830000h: success / 00000000h: failure
EDX(HI) = reply id
EDX(LO) = 0000h

DESCRIPTION
Receives RPC reply length. The reply id returned in the high order word of EDX is required to receive RPC reply data.

Note that any RPC command reached VMware application returns at least 2 bytes of reply data ("1 " on success or "0 " on failure). Even if VMware does not recognise the RPC command it returns "0 Unknown command" as the reply.

TOP

RPC subcommand 04h - Receive RPC reply data

CALL
EAX = 564D5868h - magic number
EBX = reply id from subcommand 03h
ECX(HI) = 0004h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = 4 bytes from the reply data (the first byte in LSB)
ECX = 00010000h: success / 00000000h: failure
EDX(HI) = unchanged
EDX(LO) = 0000h

DESCRIPTION
Receives the reply data from the last RPC command by 4 bytes at a time.
The first 2 bytes of the reply data always indicate the RPC command status: "1 " on success and "0 " on failure.
For example, the reply data of "machine.id.get" is "1 <*virtual machine id*>" if successfull and "0 No machine id" otherwise.

TOP

RPC subcommand 05h - Finish receiving RPC reply

CALL
EAX = 564D5868h - magic number
EBX = reply id from subcommand 03h
ECX(HI) = 0005h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = 00010000h: success / 00000000h: failure
EDX = unchanged

DESCRIPTION
I'm not really sure about the function of this command, but apparently official vmware tools invoke this command after receiving RPC reply, and since it takes reply id as one of the parameters, and it fails if you have not received the entire reply data, I decided it is somehow related to reply receiving protocol.

TOP

RPC subcommand 06h - Close RPC channel

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = 0006h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number

RETURN
EAX = ?
EBX = unchanged
ECX = 00010000h: success / 00000000h: failure
EDX = unchanged

DESCRIPTION
Close an RPC channel.

TOP

Enhanced RPC

AVAILABILITY
WS4.0 WS4.5 WS5.x GSX3.2

DESCRIPTION
The enhanced RPC mechanism introduced with WS4.0/GSX3.2 is invoked in the following steps:

- open channel with RPC subcommand 00h
- send command length with RPC subcommand 01h
- send command data through an alternate port
- receive reply length with RPC subcommand 03h
- receive reply data through an alternate port
- finish receiving reply with RPC subcommand 05h
- close channel with RPC subcommand 06h

TOP

Enhanced RPC - open channel

CALL
EAX = 564D5868h - magic number
EBX = C9435052h - enhanced RPC open magic number ('RPCI' | 80000000h)
ECX(HI) = 0000h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = don't care
EDX(LO) = 5658h - port number
ESI = don't care
EDI = don't care
EBP = don't care

RETURN
EAX = ?
EBX = unchanged
ECX = 00010000h: success / 00000000h: failure
EDX(HI) = RPC channel number
EDX(LO) = 0000h
ESI = cookie#1
EDI = cookie#2
EBP = unchanged

DESCRIPTION
Opens an RPC channel for enhanced RPC mechanism. Subsequent RPC operations require the channel number and cookies returned by this command.

TOP

Enhanced RPC - Send command length

CALL
EAX = 564D5868h - magic number
EBX = command length (not including the terminating NULL)
ECX(HI) = 0001h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
ESI = cookie#1
EDI = cookie#2
EBP = don't care

RETURN
EAX = ?
EBX = unchanged
ECX = 00810000h: success / 00000000h: failure
EDX = unchanged
ESI = unchanged
EDI = unchanged
EBP = unchanged

DESCRIPTION
Send RPC command length.

TOP

Enhanced RPC - Send command data

CALL
EAX = 564D5868h - magic number
EBX = 00010000h - data transfer magic number?
ECX = command data length (must be the same as the length sent with

subcommand 02h)
EDX(HI) = channel number
EDX(LO) = 5659h - port number (**Not the usual backdoor port!!**)
ESI = pointer to the data buffer
EDI = cookie#2
EBP = cookie#1

RETURN
EAX = unchanged
EBX = 00010000h: success / 00000000h: failure
ECX = 0
EDX = unchanged
ESI = points to the end of data buffer
EDI = unchanged
EBP = unchanged

DESCRIPTION
This is where the enhanced RPC differs notably from the legacy method. Instead of repeating *IN* instructions, whole data is transferred by a single issuing of the following instruction:

```
/* Intel syntax */
cld
rep outs dx,esi

/* AT&T syntax */
cld
rep outsb (%esi),(%dx)
```

Change of values in *ECX* and *ESI* is no more than a side effect of the above instructions.

The RPC channel must be opened for enhanced RPC.

TOP

Enhanced RPC - Recieve reply length

CALL
EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = 0003h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
ESI = cookie#1
EDI = cookie#2
EBP = don't care

RETURN
EAX = ?

EBX = reply length (not including the terminating NULL)
ECX = 00830000h: success / 00000000h: failure
EDX(HI) = reply id
EDX(LO) = 0000h
ESI = unchanged
EDI = unchanged
EBP = unchanged

DESCRIPTION
Receives RPC reply length. The reply id is returned in the high order word of *EDX* as legacy RPC but it is not used to receive reply data.

TOP

Enhanced RPC - Receive reply data

CALL
EAX = 564D5868h - magic number
EBX = 00010000h - data transfer magic number?
ECX = reply data length (must be the same as the length received with the subcommand 03h)
EDX(HI) = channel number
EDX(LO) = 5659h - port number (**Not the usual backdoor port!!**)
ESI = cookie#1
EDI = pointer to the data buffer
EBP = cookie#2

RETURN
EAX = unchanged
EBX = 00010000h: success / 00000000h: failure
ECX = 0
EDX = unchanged
ESI = unchanged
EDI = points to the end of data buffer
EBP = unchanged

DESCRIPTION
Like sending command data, this function is invoked by the following instruction:

```
/* Intel syntax */
cld
rep ins edi,dx

/* AT&T syntax */
cld
rep insb (%dx),(%edi)
```

Change of values in *ECX* and *EDI* is no more than a side effect of the instructions above.

The RPC channel must be opened for enhanced RPC.

TOP

Enhanced RPC - Finish receiving reply

CALL

EAX = 564D5868h - magic number
EBX = reply id from subcommand 03h
ECX(HI) = 0005h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
ESI = cookie#1
EDI = cookie#2
EBP = don't care

RETURN

EAX = ?
EBX = unchanged
ECX = 00010000h: success / 00000000h: failure
EDX = unchanged
ESI = unchanged
EDI = unchanged
EBP = unchanged

DESCRIPTION

See description of RPC subcommand 05h.

TOP

Enhanced RPC - Close channel

CALL

EAX = 564D5868h - magic number
EBX = don't care
ECX(HI) = 0006h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
ESI = cookie#1
EDI = cookie#2
EBP = don't care

RETURN

EAX = ?
EBX = unchanged
ECX = 00010000h: success / 00000000h: failure
EDX = unchanged
ESI = unchanged

EDI = unchanged
EBP = unchanged

DESCRIPTION

Close an enhanced RPC channel.

TOP

VM Back
Virtualization Tricks and Tools

PS. I would really appreciate it if you would let me know if you have found anything wrong with the English in this page. I'm still learning.

Comments