

Debugging Virtual Machines with the Checkpoint to Core Tool

Workstation 7.1, VMware Fusion 3.1, and ESX/ESXi 4.x

VMware now supports the `vmss2core` tool, which developers can use to debug guest operating systems and applications by converting a virtual machine checkpoint into a core dump file. The checkpoint may be either a snapshot or suspend file. You can select a variety of core dump formats that standard debuggers understand.

Compared to other debugging methods, a major advantage of `vmss2core` is that it requires absolutely no modifications to virtual machines: no additional software to install, no Windows registry keys to change, no Linux kernel modules to configure.

In practice `vmss2core` can be tremendously helpful, because it offers a debugging alternative when others fail. Most production systems run with debugging tools disabled, so when Windows produces a blue screen, or the Linux kernel panics, or the system just stops responding, there is essentially nothing you can do. Although both Windows and Linux provide extensive debugging tools, it is not possible to install them retroactively. But with `vmss2core`, after a system stops responding, you are able to create a checkpoint and immediately use a debugger to discover what the guest system was doing when it crashed.

Using the Checkpoint to Core Tool

With Workstation installed on a Windows host, you can find the `vmss2core` tool in the following directory. You might want to add this folder to your Path environment.

```
C:\Program Files\VMware\VMware Workstation
```

With Workstation installed on Linux, the `vmss2core` tool is in a standard location.

```
/usr/bin/vmss2core
```

With VMware Fusion installed on Mac OS, use the following path:

```
/Library/Application Support/VMware Fusion/vmss2core
```

Although it is available with Workstation 7.1 and Fusion 3.1, `vmss2core` is capable of converting snapshots and suspend files created by other earlier VMware products. For example, you can use the `vmss2core` tool to convert checkpoints taken of virtual machines running on ESX/ESXi hosts. As discussed below, on ESX/ESXi the `vm-support` script produces a checkpoint file, which `vmss2core` converts into a debug image.

To start debugging using `vmss2core`

- 1 Create a snapshot or suspend the virtual machine.
- 2 Locate the snapshot (`.vmsn`) or suspend file (`.vms`) in the virtual machine directory. The `vmss2core` tool also accepts monolithic or non-monolithic memory (`.vmem`) files.

If the virtual machine is not on Workstation 7.1 or Fusion 3.1, copy the checkpoint file to a Workstation 7.1 or Fusion 3.1 host for debugging.

- 3 Run the `vmss2core` tool with options selecting your preferred output type. Without any options, the tool generates one `vmss.core<N>` file (with a linear view of memory) per virtual CPU.

```
vmss2core <vmName>.vmss
```

More typically, you would use the tool with OS-specific options. For example, this command generates a `memory.dmp` file for the Windows debugger, WinDbg.

```
vmss2core -W <vmName>.vmss
```

- 4 Use an appropriate debugger to examine the core dump.

Possible debuggers include WinDbg, the Red Hat crash utility, and `gdb`.

Option Flags for Checkpoint to Core Tool

The `vmss2core` tool can produce core dump files for the Windows debugger (WinDbg), Red Hat crash compatible core files, a physical memory view suitable for the Gnu debugger `gdb`, and Mac OS X formats. [Table 1](#) documents the available option flags.

Table 1. Options of `vmss2core`

Option	Explanation
<i>(none)</i>	Without any options, produces linear views of memory (<code>vmss.core<n></code>) one per virtual CPU.
<code>-W</code>	Creates a WinDbg file (<code>memory.dmp</code>) of a Windows virtual machine with commonly used build numbers, 2195 for Win32 and 6000 for Win64.
<code>-W<num></code>	Creates a WinDbg file (<code>memory.dmp</code>) with <code><num></code> as the build number, for example: <code>-W2600</code>
<code>-WDDb<num></code>	Creates a WinDbg file (<code>memory.dmp</code>) with <code><num></code> as the debugger data block address in hexadecimal, for example: <code>-W12ac34de</code>
<code>-WSCAN</code>	Creates a WinDbg file (<code>memory.dmp</code>) and scan all of memory for the debugger data block, instead of just the lower 256 MB.
<code>-M</code>	Creates a core file (<code>vmss.core</code>) with a physical memory view suitable for the Gnu debugger <code>gdb</code> .
<code>-l <str></code>	Specifies the starting and ending offsets of Linux kernel data structures for use by the <code>-N</code> and <code>-P</code> options, with <code><str></code> expressed as <code>0xHEXNUM,0xHEXNUM</code> . Ignored when used with other options. You can determine the hexadecimal values in <code><str></code> by running the <code>getlinuxoffsets</code> debugger script. Source code for the script was posted to stackframe.blogspot.com in October 2007.
<code>-N</code>	Red Hat crash core file (<code>vmss.core</code>) for an arbitrary Linux version as defined by the <code>-l</code> option.
<code>-N4</code>	Red Hat crash core file (<code>vmss.core</code>) for Linux kernel version 2.4.
<code>-N6</code>	Red Hat crash core file (<code>vmss.core</code>) for Linux kernel version 2.6.
<code>-P</code>	Prints a list of processes running in the Linux virtual machine at checkpoint time.
<code>-P<pid></code>	Creates a core file (<code>core.<pid></code>) for the Linux process number <code><pid></code> . It is likely that programs compiled with symbol tables (not removed) will yield better debug information.
<code>-X<nn-v></code>	Mac OS core dump with <code><nn-v></code> representing architecture and Darwin kernel version.
<code>-q</code>	Quiet operation.

For Linux virtual machines, the `vmss2core` tool must guess the kernel build number and the location of debugging-related data structures. Different options control the guessing algorithms. For crash-compatible output, either specify the kernel family using the `-N4` or `-N6` options, or use a combination of `-N` and `-l <str>`. You can generate the argument for `-l` using the `getlinuxoffsets` script mentioned above.

For Windows virtual machines, the `-W` option works most of the time. If WinDbg is not completely happy with the resulting file, you might want to supply more information, for example a build number with `-W<num>`, or location of the debug block using `-WDDb<num>`.

For Mac OS virtual machines, the options `-X32` (for the 32-bit kernel) and `-X64` (for the 64-bit kernel, also called K64) should always work. You can use the options `-X32-<v>` and `-X64-<v>` if you have an exact kernel version (obtained with `uname -v`) that `vmss2core` knows about. Use of the `<v>` arguments is not recommended because this might be deprecated in the future.

The `vmss2core` tool uses only part of the checkpoint as its input. For virtual machines on desktop (hosted) products, it uses a pair of files: the `.vmss` or `.vmsn` file for virtual device information, and the `.vmem` file for the memory image. For virtual machines on ESX/ESXi hosts, it uses the `.vmss` file.

On ESX/ESXi hosts, the `vm-support` script provides the `-Z` option to suspend a virtual machine, and the `-X` option to extract a hung virtual machine. Both options take the “world ID” (available in the vSphere Client) and package the `.vmss` file into a support archive. Use the `vmss2core` tool to convert this `.vmss` checkpoint into an appropriate format for debugging.

Other Uses of Checkpoint Core Dumps

The main use of `vmss2core` is to help VMware customers diagnose problems with the guest operating system and applications running on virtual machines.

The `vmss2core` tool is good for forensics. Because it changes nothing in the virtual machine, end users inside the virtual machine cannot detect that analysis was performed. Possibly this makes `vmss2core` an interesting tool for the anti-virus and anti-malware community, especially when combined with collection honeypots based on VMware virtual machines.

The `vmss2core` tool is also good for providers of virtual appliances. Instead of building remote debugging capabilities, providers can request a compressed checkpoint of the virtual machine for analysis.

In summary, advantages of this tool are the following:

- Zero configuration. The `vmss2core` tool does not depend on swap partitions, drivers, applications, or registry settings in the guest operating system.
- Non-intrusive. You can create a snapshot of the virtual machine at any time and analyze it without interfering with guest operation.
- Fault-immune. The tool works even when the guest operating system experiences a hard hang, such as a Windows blue screen, or a Linux kernel panic with interrupts disabled.

If you have questions or want to read answers by others, go to the VMware developer forum for replay-based debugging at <http://communities.vmware.com/community/vmtn/general/guestdebugmonitor>.

If you have comments about this documentation, submit your feedback to: docfeedback@vmware.com

VMware, Inc. 3401 Hillview Ave., Palo Alto, CA 94304 www.vmware.com

Copyright © 2010 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at <http://www.vmware.com/go/patents>. VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

Item: EN-000395-00
