Important Information

Latest Version
The latest version of this document is at: http://supportcontent.checkpoint.com/documentation_download?ID=10906
For additional technical information visit Check Point Support Center (http://supportcenter.checkpoint.com).

Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 July 2010</td>
<td>Initial version</td>
</tr>
</tbody>
</table>

Feedback
Check Point is engaged in a continuous effort to improve its documentation.
Please help us by sending your comments (mailto:cp_techpub_feedback@checkpoint.com?subject=Feedback on How to Identify and Troubleshoot Memory Leaks on IPSO ).
Objective

This document explains how to identify and troubleshoot memory leaks on IPSO systems.

Supported Versions

This document was developed for Check Point R70. However, it applies to all versions of the Check Point firewall.

Supported OS

- IPSO 6.2 MR1 (GA029a02)
- IPSO 6.2 MR2 (GA039)
- The same concepts directly translate to any POSIX-compliant system

Supported Appliances

All IP Series appliances running IPSO 6.2 and Check Point firewall.
Before You Start

Related Documentation and Assumed Knowledge

- A fundamental knowledge of Unix memory management is recommended.
- Check Point Technical Assistance Center (TAC) assistance is not required to interpret the output of the `ps`, `top`, or `vmstat` commands. These are POSIX compliant commands present on all Unix systems, therefore the customer administrator is expected to have sufficient knowledge to run these tools and understand the output.
- This base knowledge is not strictly required to perform the steps outlined in this document, however the results will be confusing to people without the base level of knowledge. Therefore the actual analysis of the generated data should be performed by Check Point support.

  **Note** - The information in this document will not help resolve the issue, only help identify or prove that an issue exists.

Impact on the Environment and Warnings

**Important** - This document explains how to trace memory leaks, but in ANY procedure to trace memory leaks, there will be a significant impact on CPU resources. None of the steps outlined should be performed during core business hours on mission-critical high performance systems.
Background on Unix Memory management

Zones
The IPSO kernel manages several pools of memory that can be allocated to either kernel functions or processes. The memory pools can be viewed with `vmstat -z`, which displays all of the Universal Memory Allocator zones. Each zone is constructed with a distinctive name, and statistics about the usage:

- Zone references by slab children,
- Memory usage of the zone,
- Number of total requests in the zone.

slabs
There are children memory pools which are directly connected the zones, either by the kernel or running processes. They are called slabs. Every slab is a child of a UMA memory zone; the zone sets certain size restrictions on the type of memory pages that may be requested. The list of memory slabs may be viewed with `vmstat -m`. The reason to use memory slabs, is simply for efficiency. It is a much faster operation to create a slab of memory as a child of a pre-initialized UMA zone, than to create a memory zone from scratch. The UMA memory zones are children of a "primitive" structure called a Keg (outside the scope of this document, see references). This hierarchy of memory allocation is relevant because it can help you track down when a kernel virtual memory zone’s used pages count is increasing and never going down. This is the definition of a memory leak within the kernel.

Memory Pages
Memory within the children slabs or UMA zones is allocated on a page basis by the virtual memory manager. The size of these memory pages is fixed within the system, usually at 4kb. The IPSO memory allocator works by maintaining a set of lists that are ordered by increasing powers of two. Each list contains a set of memory blocks of its corresponding size. To fulfill a memory request, the size of the request is rounded up to the next power of two. A piece of memory is then removed from the list corresponding to the specified power of two and returned to the requester.

Thus, a request for a block of memory of size 53 returns a block from the 64-sized list. This sizing less than 4kb occurs within the 4kb slices that are allocated to a page; for example, consider that a memory operation is happening within a particular slab, and the requesting function needs 3 slots of memory, two for 512 bytes, and one for 2048 bytes. Since all of this can fit within a single virtual memory (VM) page, a single page could be used to store the data. The single VM page is allocated to a single requesting memory slab.

Allocation Chain
It is a critical requirement that memory allocation occur quickly. In IPSO, there are 3 structures that are defined which manage individual memory pages. Chaining together the structures allows a program to use predefined memory types, sizes and meta-information. This speeds up the allocation substantially and increases performance.

1. Keg – Provides structural information about the memory allocation
2. UMA Zone – Inherits information from a certain Keg structure, and further defines sizing and key attributes
3. Memory slab – Inherits information from one or more UMA zones, and is linked to directly by the requesting task

Virtual Memory
Any modern operating system uses a Virtual Memory system to allocate memory space to the kernel and applications. This virtual system maps memory requests to contiguous address space, which maps to discontiguous physical page locations. The virtual memory handler is used to prevent resource exhaustion of physical memory. By tracking memory accesses and allocation/free operations, it is able to dynamically free up physical memory as needed by remapping a physical memory page to the disk media – this is called...
swapping. Swapping of memory is normal and happens as part of memory management. However during standard operating conditions, a Check Point firewall should NOT swap as this would lead to severely degraded performance.

**Wired Memory**

A portion of the physical memory can be reserved as WIRED memory. Wired memory refers to memory pages that may not be swapped out of physical memory to disk. Often the main usage of Wired memory is for core kernel memory structures. Programs running on the system may request a certain percentage of their memory space to be reserved as Wired memory. The total memory allocations and usage may be viewed using the "top" command.
How to Use This Document

Most often this document will be used because there is a suspected memory leak on a system. The reason for assuming there is a memory leak, is typically that the system crashed. A system could crash because there was insufficient Free memory, and no other type of memory could be Free’d (possibly too much was Wired, or the Virtual Memory system was exhausted), and a new memory allocation for a core system function failed. In this instance it is normal for the system to crash or lock up.

After the system has been recovered with a hard power cycle, there are very few clues about what originally caused the problem. The logfiles most often could not be written to, because there was no memory available to initialize the function that would write the log. All crucial IPSO kernel counters available via ipsctl would be cleared on a system reboot. Finally, the "top" "vmstat" and "ps" output would also be cleared.

In the case where there is no available core file to analyze, an analysis must be done on the newly-booted system to determine if the problem is persistent and likely to occur again. Memory leaks are notoriously difficult to track down.

The Memory Leak Detection Script

The script accompanying this document is intended to help trace memory leaks and determine if there is indeed a memory leak present, so that the Check Point development team may be engaged to try and help narrow down the exact cause. The script by itself will only serve as instrumentation.

If the script is aborted or the system crashes before the script is done, the raw stats will still be present but the script must be hacked (by removing the data collection portion and all sections above it) and rerun to generate the final output.

The script is included in a package that also includes a sample output showing the script running on IPSO 6.2 MR2 (GA039).
Running the Memory Leak Detection Script

1. Download the memory leak detection script mem-html.sh (http://supportcontent.checkpoint.com/file_download?id=10918), and copy the script to the target system. The script is included in a package that also includes a sample output showing the script running on IPSO 6.2 MR2 (GA039).

2. You may wish to read the script before executing it.

   **Important** - The script is VERY CPU INTENSIVE and may conceivably cause traffic loss on production systems.

3. For typical operations it is recommended to run the script with the following syntax:
   ```
   sh mem-html.sh 3600
   ```
   This will execute the script, and run for approximately one hour – 3600 seconds.
   If you are concerned about the CPU utilization of the script, you may run it with the following syntax:
   ```
   nice +20 sh mem-html.sh 3600
   ```
   The script will be executed with the lowest possible CPU priority on the system; however the results may be less accurate.

The script works by writing an HTML page and several subdirectories containing raw statistics. The raw stats are compiled and written in an abbreviated format to the HTML page as graphs and tables.

**Script output, collected before the intensive data collection**

The script will collect several important data and include them at the top of the document.

![Check Point Logo]

This script was started for 400 seconds.

**System uname**: IPSO ipso6-vm-instructor 6.2-GA029a02 releng 1 12.17.2009-205558 i386

**Uptime is** 11:31AM up 2 days, 18:15, 2 users, load averages: 0.00, 0.01, 0.00 at script start

**Script started** Mon Mar 1 11:31:49 EST 2010

---

**Top process utilization before stats collected:**

last pid: 3468; load averages: 0.00, 0.01, 0.00 up 2+18:15:51  11:31:49
60 processes: 6 running, 44 sleeping, 14 waiting

**Mem**: 19M Active, 158M Inact, 123M Wired, 24K Cachex, 99M Buf, 1691M Free
Swap: 2439M Total, 2439M Free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>root</td>
<td>171</td>
<td>52</td>
<td>0K</td>
<td>16K</td>
<td>RUN</td>
<td>55.1%</td>
<td>96.78%</td>
<td>cpu0</td>
</tr>
<tr>
<td>478</td>
<td>root</td>
<td>44</td>
<td>0</td>
<td>1992K</td>
<td>1496K</td>
<td>select</td>
<td>27:24</td>
<td>0.30%</td>
<td>ifm</td>
</tr>
</tbody>
</table>
The basic information which is collected includes the duration of the script run, the kernel version, uptime, and date when the script was executed. The “top” output is also collected for later reference.

**VMStat outputs at script start**

```
vmstat -m before
vmstat -z before
vmstat 1 5 before
vmstat -s before
```

The vmstat outputs listed are collected before, and after, the script data collection.

**Intensive data collection**

Until the end timer is reached, the script will iterate through data collection using ps and top. This output is collected in the appropriate subdirectory.

```
ips06-vm-instructor[admin]# sh mem-html.sh 10
This script will record detailed memory statistics, to try and track memory leaks
using inbuilt IP50 tools. Please do not interrupt. The script will run for around 10 seconds
*** The script uses some additional time for setup and cleanup ***

Iteration 1, there are around 6 seconds left until this script terminates.
We are done the data collection, please wait...
*** PLEASE IGNORE ERROR MESSAGES RELATED TO EXPR ***
expr: syntax error
[ : -eq: unexpected operator
expr: syntax error
DONE!
Please provide Technical Support the script-output.tgz file.
ips06-vm-instructor[admin]#
```

**Wrapping up the data**

Once the script's main loop is complete, the data is graphed within the HTML page using CSS. This is to assist the analyst by providing a historical plot of the memory utilization in megabytes.
Tables are also created containing the start and end values of both kernel memory stats, and process memory stats. This allows the tracking of memory leaks both within the kernel space, as well as userland.

<table>
<thead>
<tr>
<th>Memory type</th>
<th>Start value (mb) at</th>
<th>End value (mb) at</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>145</td>
<td>11</td>
</tr>
<tr>
<td>Inactive</td>
<td>57</td>
<td>58</td>
</tr>
<tr>
<td>Wired</td>
<td>1486</td>
<td>2308</td>
</tr>
<tr>
<td>Cache</td>
<td>28K</td>
<td>118</td>
</tr>
<tr>
<td>Buffer</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Free</td>
<td>811</td>
<td>4684K</td>
</tr>
</tbody>
</table>

A leak in the kernel should be easily detectable by seeing the Free memory consistently shrink, and Wired memory growing consistently. There should be a corresponding growing Virtual Memory slab corresponding to the system function with the leak. Finally, the RSS for every process would drop as the virtual memory system tries to reclaim the least frequently used memory pages from the running processes.

A leak in a userland process would show similar behavior, except there would be a large increase in RSS and VSZ on one specific process. The system "should" normally panic in these circumstances either when the process reserved Wired memory increases beyond the available Free space, or the Virtual Memory address space is exhausted. It has been observed that a system may simply hang rather than panic and generate a core file.

As a convenience to the analyst, the file descriptor allocations are also tracked via the script, since file descriptor leaks could also be interpreted as a memory leak.

All of the raw data that is used by the script is stored in the subdirectories so that the analyst may drop the results into Excel and plot them.

The number of open file descriptors was 209 when we started, and was 191 when we finished this script.

All of the raw data that is used by the script is stored in the subdirectories so that the analyst may drop the results into Excel and plot them.

### Script output

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td></td>
<td>Folder</td>
</tr>
<tr>
<td>pstat</td>
<td></td>
<td>Folder</td>
</tr>
<tr>
<td>run-script</td>
<td></td>
<td>Folder</td>
</tr>
<tr>
<td>securexd</td>
<td></td>
<td>Folder</td>
</tr>
<tr>
<td>top-pdps</td>
<td></td>
<td>Folder</td>
</tr>
<tr>
<td>vmstat</td>
<td></td>
<td>Folder</td>
</tr>
<tr>
<td>index.html</td>
<td>58 KB</td>
<td>Firefox Document</td>
</tr>
</tbody>
</table>

The resulting index.html and subdirectories are put into a script-output.tgz tarball, and the original files and directories are then deleted. The script-output.tgz must be collected and provided to Check Point TAC for analysis. Check Point will extract the contents of the tarball.
Analysis of the Memory Leak Detection script

The analyst of the script output must consider several items:

- How busy is the system?
  - The system may only leak memory when there is a lot of traffic
  - Running the script during a maintenance window may not yield the desired output
- How long it has been up?
  - A system could leak memory over very long timeframes – potentially weeks or months
- Has the memory gone up or down normally, or abnormally?
  - A system will during a normal day request and release large amounts of memory
  - "Abnormal" growth could indicate the problem, but it must be considered with everything else that is happening
- Where is the memory going?
  - Depending on the uptime and how busy the system is, memory can be allocated to Wired, Inactive and process memory space in a way that looks wrong
  - This is dependent on too many factors to worry about any one indicator
- Has the condition that triggered the memory leak occurred?
  - A memory leak is usually triggered by a specific function that requests memory and never releases it. If this function doesn’t execute, memory will not leak.
- It’s important to run the script on the actual system that is leaking memory, or a proven replication

When running the script, ensure that an appropriate duration is selected. Unless the memory leak is extreme, running the script for hours, days or weeks may be required.
Additional Tools

Commands
The following commands may prove useful when trying to track down memory leaks:

```bash
top
ps -auxwwlSHm
vmstat -z
vmstat -m
vmstat 1 5
vmstat -s
top -mio -S -H
```

Kernel Flags
IPSO is compiled with an additional debug kernel. Since IPSO is FreeBSD-based, you can look up additional information about what the kernel flags are for. The additional flags that the kernel is compiled with, are

- Witness
- Witness_KDB
- Witness_Skipspin
- Invariants
- Invariants_Support
- Diagnostic
- KDB_Unattended

Choosing to run this `kernel.debug` is extremely unwise unless directed to do so by Check Point engineering. One purpose of this debug kernel is to crash/panic when an error is detected, much more frequently than on a non-debug kernel. Therefore the kernel is very rarely provided to a customer and would rather be implemented in a lab where the error occurred.

References
The following resources are good starting places to learn about memory management:

- The Slab Allocator: An Object-Caching Kernel Memory Allocator (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.4759&rep=rep1&type=pdf)
- Introduction to Debugging the FreeBSD Kernel (http://www.bsdcan.org/2008/schedule/attachments/45_article.pdf)
- Week 3 of Berkeley CS61C Spring 2010 Lectures (http://inst.eecs.berkeley.edu/~cs61c/sp10/)
- Computer Science 152: Computer Architecture and Engineering Lectures, Berkeley University (http://inst.eecs.berkeley.edu/~cs152/sp10/)
- Network Buffer Allocation in the FreeBSD Operating System, Bosko Milekic (http://bmilekic.unixdaemons.com/netbuf_bmilekic.pdf)