Extending the linux_slabinfo Volatility plugin

Fulvio Di Girolamo - Angelo Russi
EURECOM

July 22, 2021

1 The Project

The objective of our project was the extension of the linux_slabinfo plugin of the Volatility Framework in order to provide statistics about the slab allocator for Linux systems using either SLOB or SLUB.

Due to its simplicity we started out by developing the functionality for SLOB analysis, using dumps and profiles for systems running version 5.7.2 of the Linux kernel, then we focused on SLUB. In this second phase of our work we found ourselves forced to test our code with dumps and profiles from a much older version of the kernel (3.5.7), because when using more recent versions the Volatility profile we extracted from the system did not contain the definition of some vital data structures such as either *kmem_cache* or *kmem_cache_node*.

We think it is also worth mentioning that due to a general lack of documentation for both SLOB and SLUB as well as for the API of Volatility we had to do our best to extract information directly from their source codes, which may have been the cause of at least some of the difficulties we encountered during the execution of the project.

The following sections present a more thorough description of how the plugin works and the problems we encountered during development.

The code for the plugin can be found at our fork of the official Volatility repository.

2 linux_slabinfo for SLOB

When using the plugin with a dump from a system using the SLOB allocator, some options can be specified by the user:

- -p, –page_size: specify the page size of the system, used to determine the size of a single SLOB_UNIT (the metric for object sizes and offsets in the allocator)
- -L, –dump_free_list: specify the linked list (free_slob_(s)mall, free_slob_(m)edium, free_slob_(l)arge, (a)ll) whose free objects should be dumped to a file

- -D, –dump_file: specify the path of the file which will contain the dump (must be used together with -L)

The output of the plugin contains various information. In particular, for each of the three linked lists, it outputs the total number of free objects in the pages of the list, divided in 4 bins of equal size depending on the maximum size accepted in the list (256 B for the "small" list, 1024 B for "medium" and the page size for "large"); for each of the lists, it also outputs some general statistics:

- The total free space in the list
- The mean size of free objects in the list
- The number of pages in the list

An example output of the plugin is in Figure 1.

```
Volatility Foundation Volatility Framework 2.6.1
<list_name>          <range (bytes)> <# free>
-------------------- --------------- ----------
free_slob_small      0-63            22338
free_slob_small      64-127          8245
free_slob_small      128-191         3550
free_slob_small      192-255         6847
------------ slob small stats -----------------
free space 3207938 | free_mean_size 78 | PAGES 7641
-------------------- --------------- ----------
free_slob_medium     0-255           4073
free_slob_medium     256-511         2555
free_slob_medium     512-767         3981
free_slob_medium     768-1023        986
------------ slob medium stats ----------------
free space 4388976 | free_mean_size 378 | PAGES 7103
-------------------- --------------- ----------
free_slob_large      0-1023          4004
free_slob_large      1024-2047       249
free_slob_large      2048-3071       90
free_slob_large      3072-4095       1
------------ slob large stats ----------------
free space 3292524 | free_mean_size 757 | PAGES 3428
```

Figure 1: The output of linux_slabinfo for SLOB

Concerning the file dump, its format is the following:

- Although a page could be in more than one free list and contain free objects of various sizes, only objects of the correct size according to the considered free list are actually dumped

- For each free object being dumped, the file contains a human-readable header with information about the object address and size, followed by the binary dump of the object itself

Notice that in order to extract this information from the dump, we needed to use the API of Volatility to parse the objects of type *page* inside each linked list and then for each page walk through its list of free objects (the first of which is pointed by the *freelist* member of the struct).

3 linux_slabinfo for SLUB

Running the plugin with dumps from systems using the SLUB allocator requires no options to be specified and provides an output with the same format as the output of **sudo cat /proc/slabinfo** on a live system.

In this case, due to the structure of the SLUB allocator extracting all the necessary information from the dump requires multiple steps:

- Retrieving all the objects of type *kmem_cache* from the *slab_caches* list
- For each cache, extract the object of type *kmem_cache_node* pointed by the member *node* of the *kmem_cache* struct
- The member *partial* of *kmem_cache_node* should point to a linked list of *page* objects, each of which should contain a pointer to the first of its free objects. Therefore, extracting the pages in this list should allow to walk through their free objects and count them as for the SLOB allocator.

Notice that at first we also tried to take into account the content of objects of type *kmem_cache_cpu* (pointed by the member *cpu_slab* of *kmem_cache*), but we had to abandon this idea as it being a per-cpu variable made it hard to identify those structures within the dump. From the limited documentation available we also inferred that the information contained in *kmem_cache_cpu* is somewhat redundant with the information contained in *kmem_cache_node*, therefore we assumed it was not necessary to put more effort in this direction.

Unfortunately, apart from being forced to use older kernel versions to test our code as mentioned above, we encountered some difficulties with correctly parsing the objects of type *page* which are supposedly contained in the linked list pointed by *partial*. Although most of the information could be successfully extracted from the *kmem_cache* and *kmem_cache_node* structures, this prevented us from being able to count the free objects in each cache, which we would have used to derive the number of active objects when different from the total number of objects.

It is worth mentioning that in order to understand the root cause of this problem we tried to simplify the system from which we extracted the dumps and profiles; in particular,

we tried to analyze dumps from monocore systems with slab merging disabled, for both 32 and 64 bit architectures, with no success. An hypotesis we made is that *partial* may point not to the *page* objects themselves but to their *lru* member of type *list_head*, but due to time constraints we were not able to verify it and had no choice but presenting the product as-is.