```
**********************************************************
   37271 Thu Jun  4 15:50:23 2015
new/usr/src/lib/libbe/common/be_list.c
5679 be_sort_list(): Possible null pointer dereference
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
  24  */

  26 /*
  27  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
  28  * Copyright 2015 Toomas Soome <tsoome@me.com>
  29  * Copyright 2015 Gary Mills
  30  */

  32 #include <assert.h>
  33 #include <libintl.h>
  34 #include <libnvpair.h>
  35 #include <libzfs.h>
  36 #include <stdio.h>
  37 #include <stdlib.h>
  38 #include <string.h>
  39 #include <strings.h>
  40 #include <sys/types.h>
  41 #include <sys/stat.h>
  42 #include <unistd.h>
  43 #include <errno.h>

  45 #include <libbe.h>
  46 #include <libbe_priv.h>

  48 /*
  49  * Callback data used for zfs_iter calls.
  50  */
  51 typedef struct list_callback_data {
  52         char *zpool_name;
  53         char *be_name;
  54         be_node_list_t *be_nodes_head;
  55         be_node_list_t *be_nodes;
  56         char current_be[MAXPATHLEN];
  57 } list_callback_data_t;
_____unchanged_portion_omitted_

 682 /*
 683  * Function:    be_sort_list
```



```
 684  * Description: Sort BE node list
 685  * Parameters:
 686  *              pointer to address of list head
 687  *              compare function
 688  * Returns:
 689  *              nothing
 690  * Side effect:
 691  *              node list sorted by name
 692  * Scope:
 693  *              Private
 694  */
 695 static void
 696 be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
 697 {
 698         size_t ibe, nbe;
 699         be_node_list_t *p = NULL;
 700         be_node_list_t **ptrlist = NULL;
 701         be_node_list_t **ptrtmp;

 703         if (pstart == NULL)
 704                 return;
 705         /* build array of linked list BE struct pointers */
 706         for (p = *pstart, nbe = 0; p != NULL; nbe++, p = p->be_next_node) {
 707                 ptrtmp = realloc(ptrlist,
 705                 ptrlist = realloc(ptrlist,
 708                     sizeof (be_node_list_t *) * (nbe + 2));
 709                 if (ptrtmp == NULL) { /* out of memory */
 710                         be_print_err(gettext("be_sort_list: memory "
 711                             "allocation failed\n"));
 712                         goto free;
 713                 }
 714                 ptrlist = ptrtmp;
 715                 ptrlist[nbe] = p;
 716         }
 717         if (nbe == 0)
 718                 return;
 719         /* in-place list quicksort using qsort(3C) */
 720         if (nbe > 1)    /* no sort if less than 2 BEs */
 721                 qsort(ptrlist, nbe, sizeof (be_node_list_t *), compar);

 723         ptrlist[nbe] = NULL; /* add linked list terminator */
 724         *pstart = ptrlist[0]; /* set new linked list header */
 725         /* for each BE in list */
 726         for (ibe = 0; ibe < nbe; ibe++) {
 727                 size_t k, ns;   /* subordinate index, count */

 729                 /* rewrite list pointer chain, including terminator */
 730                 ptrlist[ibe]->be_next_node = ptrlist[ibe + 1];
 731                 /* sort subordinate snapshots */
 732                 if (ptrlist[ibe]->be_node_num_snapshots > 1) {
 733                         const size_t nmax = ptrlist[ibe]->be_node_num_snapshots;
 734                         be_snapshot_list_t ** const slist =
 735                             malloc(sizeof (be_snapshot_list_t *) * (nmax + 1));
 736                         be_snapshot_list_t *p;

 738                         if (slist == NULL)
 739                                 continue;
 740                         /* build array of linked list snapshot struct ptrs */
 741                         for (ns = 0, p = ptrlist[ibe]->be_node_snapshots;
 742                             ns < nmax && p != NULL;
 743                             ns++, p = p->be_next_snapshot) {
 744                                 slist[ns] = p;
 745                         }
 746                         if (ns < 2)
 747                                 goto end_snapshot;
 748                         slist[ns] = NULL; /* add terminator */
```

```
749                         /* in-place list quicksort using qsort(3C) */
750                         qsort(slist, ns, sizeof (be_snapshot_list_t *),
751                             be_qsort_compare_snapshots);
752                         /* rewrite list pointer chain, including terminator */
753                         ptrlist[ibe]->be_node_snapshots = slist[0];
754                         for (k = 0; k < ns; k++)
755                                 slist[k]->be_next_snapshot = slist[k + 1];
756 end_snapshot:
757                         free(slist);
758                 }
759                 /* sort subordinate datasets */
760                 if (ptrlist[ibe]->be_node_num_datasets > 1) {
761                         const size_t nmax = ptrlist[ibe]->be_node_num_datasets;
762                         be_dataset_list_t ** const slist =
763                             malloc(sizeof (be_dataset_list_t *) * (nmax + 1));
764                         be_dataset_list_t *p;

766                         if (slist == NULL)
767                                 continue;
768                         /* build array of linked list dataset struct ptrs */
769                         for (ns = 0, p = ptrlist[ibe]->be_node_datasets;
770                             ns < nmax && p != NULL;
771                             ns++, p = p->be_next_dataset) {
772                                 slist[ns] = p;
773                         }
774                         if (ns < 2) /* subordinate datasets < 2 - no sort */
775                                 goto end_dataset;
776                         slist[ns] = NULL; /* add terminator */
777                         /* in-place list quicksort using qsort(3C) */
778                         qsort(slist, ns, sizeof (be_dataset_list_t *),
779                             be_qsort_compare_datasets);
780                         /* rewrite list pointer chain, including terminator */
781                         ptrlist[ibe]->be_node_datasets = slist[0];
782                         for (k = 0; k < ns; k++)
783                                 slist[k]->be_next_dataset = slist[k + 1];
784 end_dataset:
785                         free(slist);
786                 }
787         }
788 free:
789         free(ptrlist);
790 }
```

*unchanged_portion_omitted_*