

```

*****
4201 Wed Jun 18 15:41:36 2014
new/usr/src/cmd/pfexec/pfexec.c
4577 pfexec's error reporting is (at least sometimes) awful
*****
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 */
21 /*
22 * Copyright 2014 Gary Mills
23 * Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 /*
27 * New implementation of pfexec(1) and all of the profile shells.
28 *
29 * The algorithm is as follows:
30 *   first try to derive the shell's path from getexecname();
31 *   note that this requires a *hard* link to the program, so
32 *   if we find that we are actually executing pfexec, we start
33 *   looking at argv[0].
34 *   argv[0] is also our fallback in case getexecname doesn't find it.
35 */
36 #include <sys/param.h>
37 #include <alloca.h>
38 #include <errno.h>
39 #include <locale.h>
40 #include <priv.h>
41 #include <stdio.h>
42 #include <stdlib.h>
43 #include <string.h>
44 #include <unistd.h>

46 #define PFEEXEC "pfexec"
47 #ifndef TEXT_DOMAIN
48 #define TEXT_DOMAIN "SYS_TEST"
49 #endif

51 #define RES_PFEEXEC 1
52 #define RES_OK 0
53 #define RES_FAILURE -1

55 /*
56 * Return the shellname
57 */
58 int
59 shellname(const char *name, char buf[MAXPATHLEN])
60 {
61     const char *cmd = strrchr(name, '/');

```

```

63     if (cmd == NULL)
64         cmd = name;
65     else
66         cmd++;

68     if (strncmp(cmd, "pf", 2) != 0)
69         return (RES_FAILURE);

71     if (strcmp(cmd, PFEEXEC) == 0)
72         return (RES_PFEEXEC);

74     if (strlen(name) >= MAXPATHLEN)
75         return (RES_FAILURE);

77     if (cmd == name) {
78         (void) strcpy(buf, cmd + 2, MAXPATHLEN);
79     } else {
80         (void) strncpy(buf, name, cmd - name);
81         (void) strcpy(buf + (cmd - name), cmd + 2);
82     }
83     return (RES_OK);

85 }
    unchanged portion omitted

94 int
95 main(int argc, char **argv)
96 {
97     char *cmd;
98     char *pset = NULL;
99     char pathbuf[MAXPATHLEN];
100     int c;
101     priv_set_t *wanted;
102     int oflag;

104     oflag = getpflags(PRIV_PFEEXEC);
105     if (setpflags(PRIV_PFEEXEC, 1) != 0) {
106         (void) fprintf(stderr,
107             "pfexec: unable to set PFEEXEC flag: %s\n",
108             strerror(errno));
109         perror("setpflags(PRIV_PFEEXEC)");
110         exit(1);
111     }

112     if (*argv[0] == '-')
113         cmd = argv[0] + 1;
114     else
115         cmd = argv[0];

117     /* Strip "pf" from argv[0], it confuses some shells. */
118     if (strncmp(cmd, "pf", 2) == 0) {
119         argv[0] += 2;
120         /* argv[0] will need to start with '-' again. */
121         if (argv[0][-2] == '-')
122             *argv[0] = '-';
123     }

125     /* If this fails, we just continue with plan B */
126     if (shellname(getexecname(), pathbuf) == RES_OK)
127         (void) execv(pathbuf, argv);

129     switch (shellname(cmd, pathbuf)) {
130     case RES_OK:
131         (void) execv(pathbuf, argv);
132         (void) fprintf(stderr,

```

```
133     gettext("pfexec: unable to execute %s: %s\n"),
134     pathbuf, strerror(errno));
129     perror(pathbuf);
135     return (1);
136 case RES_PFEEXEC:
137 case RES_FAILURE:
138     while ((c = getopt(argc, argv, "P:")) != EOF) {
139         switch (c) {
140             case 'P':
141                 if (pset == NULL) {
142                     pset = optarg;
143                     break;
144                 }
145                 /* FALLTHROUGH */
146             default:
147                 usage();
148         }
149     }
150     argc -= optind;
151     argv += optind;
152     if (argc < 1)
153         usage();
155     if (pset != NULL) {
156         if ((wanted = priv_str_to_set(pset, ",", NULL)) ==
157             NULL) {
158             (void) fprintf(stderr,
159                 gettext("pfexec: error parsing "
160                     "privileges: %s\n"), strerror(errno));
161             exit(EXIT_FAILURE);
162         }
163         wanted = priv_str_to_set(pset, ",", NULL);
164         if (setppriv(PRIV_ON, PRIV_INHERITABLE, wanted) != 0) {
165             (void) fprintf(stderr,
166                 gettext("pfexec: error setting "
167                     "privileges: %s\n"), strerror(errno));
168             gettext("setppriv(): %s\n"),
169                 strerror(errno);
170             exit(EXIT_FAILURE);
171         }
172         (void) setpflags(PRIV_PFEEXEC, oflag);
173     }
174     (void) execvp(argv[0], argv);
175     (void) fprintf(stderr,
176         gettext("pfexec: unable to execute %s: %s\n"),
177         argv[0], strerror(errno));
178     perror(argv[0]);
179     return (1);
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
```

unchanged portion omitted