****
4999 Mon Jul 23 18:44:43 2012
new/usr/src/cmd/boot/common/mboot_extra.c
3027 installgrub can segfault when encountering bogus data on disk
****
_____***unchanged_portion_omitted_***

```
 105 /*
 106  * Given a pointer to the extra information area (a sequence of bb_header_ext_t
 107  * + payload chunks), find the extended information structure.
 108  */
 109 bblk_einfo_t *
 110 find_einfo(char *extra, uint32_t size)
 110 find_einfo(char *extra)
 111 {
 112         bb_header_ext_t         *ext_header;
 113         bblk_einfo_t            *einfo;
 114         uint32_t                cksum;

 116         assert(extra != NULL);

 118         ext_header = (bb_header_ext_t *)extra;
 119         if (ext_header->size > size) {
 120                 BOOT_DEBUG("Unable to find extended versioning information, "
 121                     "data size too big\n");
 122                 return (NULL);
 123         }

 125         cksum = compute_checksum(extra + sizeof (bb_header_ext_t),
 126             ext_header->size);
 127         BOOT_DEBUG("Extended information header checksum is %x\n", cksum);

 129         if (cksum != ext_header->checksum) {
 130                 BOOT_DEBUG("Unable to find extended versioning information, "
 131                     "data looks corrupted\n");
 132                 return (NULL);
 133         }

 135         /*
 136          * Currently we only have one extra header so it must be encapsulating
 137          * the extended information structure.
 138          */
 139         einfo = (bblk_einfo_t *)(extra + sizeof (bb_header_ext_t));
 140         if (memcmp(einfo->magic, EINFO_MAGIC, EINFO_MAGIC_SIZE) != 0) {
 141                 BOOT_DEBUG("Unable to read stage2 extended versioning "
 142                     "information, wrong magic identifier\n");
 143                 BOOT_DEBUG("Found %s, expected %s\n", einfo->magic,
 144                     EINFO_MAGIC);
 145                 return (NULL);
 146         }

 148         return (einfo);
 149 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1709 Mon Jul 23 18:44:45 2012
new/usr/src/cmd/boot/common/mboot_extra.h
3027 installgrub can segfault when encountering bogus data on disk
**********************************************************
______unchanged_portion_omitted_

  48 uint32_t compute_checksum(char *, uint32_t);
  49 bblk_einfo_t *find_einfo(char *, uint32_t);
  49 bblk_einfo_t *find_einfo(char *);
  50 int find_multiboot(char *, uint32_t, uint32_t *);
  51 void add_einfo(char *, char *, bblk_hs_t *, uint32_t);
  52 int compare_bootblocks(char *, char *, char **);

  54 #ifdef  __cplusplus
  55 }
______unchanged_portion_omitted_
```

```
**********************************************************
   25366 Mon Jul 23 18:44:45 2012
new/usr/src/cmd/boot/installboot/installboot.c
3027 installgrub can segfault when encountering bogus data on disk
**********************************************************
______unchanged_portion_omitted_

 178 static int
 179 read_bootblock_from_disk(int dev_fd, ib_bootblock_t *bblock)
 180 {
 181         char                    *dest;
 182         uint32_t                size;
 183         uint32_t                buf_size;
 184         uint32_t                mboot_off;
 185         multiboot_header_t      *mboot;

 187         assert(bblock != NULL);
 188         assert(dev_fd != -1);

 190         /*
 191          * The ZFS bootblock is divided in two parts, but the fake multiboot
 192          * header can only be in the second part (the one contained in the ZFS
 193          * reserved area).
 194          */
 195         if (read_in(dev_fd, mboot_scan, sizeof (mboot_scan),
 196             BBLK_ZFS_EXTRA_OFF) != BC_SUCCESS) {
 197                 BOOT_DEBUG("Error reading ZFS reserved area\n");
 198                 perror("read");
 199                 return (BC_ERROR);
 200         }

 202         /* No multiboot means no chance of knowing bootblock size */
 203         if (find_multiboot(mboot_scan, sizeof (mboot_scan), &mboot_off)
 204             != BC_SUCCESS) {
 205                 BOOT_DEBUG("Unable to find multiboot header\n");
 206                 return (BC_NOEXTRA);
 207         }
 208         mboot = (multiboot_header_t *)(mboot_scan + mboot_off);

 210         /*
 211          * Currently, the amount of space reserved for extra information
 212          * is "fixed". We may have to scan for the terminating extra payload
 213          * in the future.
 214          */
 215         size = mboot->load_end_addr - mboot->load_addr;
 216         buf_size = P2ROUNDUP(size + SECTOR_SIZE, SECTOR_SIZE);
 217         bblock->file_size = size;

 219         bblock->buf = malloc(buf_size);
 220         if (bblock->buf == NULL) {
 221                 BOOT_DEBUG("Unable to allocate enough memory to read"
 222                     " the extra bootblock from the disk\n");
 223                 perror(gettext("Memory allocation failure"));
 224                 return (BC_ERROR);
 225         }
 226         bblock->buf_size = buf_size;

 228         dest = bblock->buf;
 229         size = BBLK_DATA_RSVD_SIZE;

 231         if (read_in(dev_fd, dest, size, SECTOR_SIZE) != BC_SUCCESS) {
 232                 BOOT_DEBUG("Error reading first %d bytes of the bootblock\n",
 233                     size);
 234                 (void) free(bblock->buf);
 235                 bblock->buf = NULL;
 236                 return (BC_ERROR);
```



```
 237         }

 239         dest += BBLK_DATA_RSVD_SIZE;
 240         size = bblock->buf_size - BBLK_DATA_RSVD_SIZE;

 242         if (read_in(dev_fd, dest, size, BBLK_ZFS_EXTRA_OFF) != BC_SUCCESS) {
 243                 BOOT_DEBUG("Error reading ZFS reserved area the second time\n");
 244                 (void) free(bblock->buf);
 245                 bblock->buf = NULL;
 246                 return (BC_ERROR);
 247         }

 249         /* Update pointers. */
 250         bblock->file = bblock->buf;
 251         bblock->mboot_off = mboot_off;
 252         bblock->mboot = (multiboot_header_t *)(bblock->buf + bblock->mboot_off
 253             + BBLK_DATA_RSVD_SIZE);
 254         bblock->extra = (char *)bblock->mboot + sizeof (multiboot_header_t);
 255         bblock->extra_size = bblock->buf_size - bblock->mboot_off
 256             - BBLK_DATA_RSVD_SIZE - sizeof (multiboot_header_t);
 257         return (BC_SUCCESS);
 258 }

 260 static boolean_t
 261 is_update_necessary(ib_data_t *data, char *updt_str)
 262 {
 263         bblk_einfo_t    *einfo;
 264         bblk_hs_t       bblock_hs;
 265         ib_bootblock_t  bblock_disk;
 266         ib_bootblock_t  *bblock_file = &data->bootblock;
 267         ib_device_t     *device = &data->device;
 268         int             dev_fd = device->fd;

 270         assert(data != NULL);
 271         assert(device->fd != -1);

 273         /* Nothing to do if we are not updating a ZFS bootblock. */
 274         if (!is_zfs(device->type))
 275                 return (B_TRUE);

 277         bzero(&bblock_disk, sizeof (ib_bootblock_t));

 279         if (read_bootblock_from_disk(dev_fd, &bblock_disk) != BC_SUCCESS) {
 280                 BOOT_DEBUG("Unable to read bootblock from %s\n", device->path);
 281                 return (B_TRUE);
 282         }

 284         einfo = find_einfo(bblock_disk.extra, bblock_disk.extra_size);
 282         einfo = find_einfo(bblock_disk.extra);
 285         if (einfo == NULL) {
 286                 BOOT_DEBUG("No extended information available\n");
 287                 return (B_TRUE);
 288         }

 290         if (!do_version || updt_str == NULL) {
 291                 (void) fprintf(stdout, "WARNING: target device %s has a "
 292                     "versioned bootblock that is going to be overwritten by a "
 293                     "non versioned one\n", device->path);
 294                 return (B_TRUE);
 295         }

 297         if (force_update) {
 298                 BOOT_DEBUG("Forcing update of %s bootblock\n", device->path);
 299                 return (B_TRUE);
 300         }
```

```
302         BOOT_DEBUG("Ready to check installed version vs %s\n", updt_str);

304         bblock_hs.src_buf = (unsigned char *)bblock_file->file;
305         bblock_hs.src_size = bblock_file->file_size;

307         return (einfo_should_update(einfo, &bblock_hs, updt_str));
308 }
_____unchanged_portion_omitted_

659 /*
660  * Retrieves from a device the extended information (einfo) associated to the
661  * installed bootblock.
662  * Expects one parameter, the device path, in the form: /dev/rdsk/c?[t?]d?s0.
663  * Returns:
664  *        - BC_SUCCESS (and prints out einfo contents depending on 'flags')
665  *        - BC_ERROR (on error)
666  *        - BC_NOEINFO (no extended information available)
667  */
668 static int
669 handle_getinfo(char *progname, char **argv)
670 {

672         ib_data_t       data;
673         ib_bootblock_t  *bblock = &data.bootblock;
674         ib_device_t     *device = &data.device;
675         bblk_einfo_t    *einfo;
676         uint8_t         flags = 0;
677         uint32_t        size;
678         char            *device_path;
679         int             retval = BC_ERROR;
680         int             ret;

682         device_path = strdup(argv[0]);
683         if (!device_path) {
684                 (void) fprintf(stderr, gettext("Missing parameter"));
685                 usage(progname);
686                 goto out;
687         }

689         bzero(&data, sizeof (ib_data_t));
690         BOOT_DEBUG("device path: %s\n", device_path);

692         if (init_device(device, device_path) != BC_SUCCESS) {
693                 (void) fprintf(stderr, gettext("Unable to gather device "
694                     "information from %s\n"), device_path);
695                 goto out_dev;
696         }

698         if (!is_zfs(device->type)) {
699                 (void) fprintf(stderr, gettext("Versioning only supported on "
700                     "ZFS\n"));
701                 goto out_dev;
702         }

704         ret = read_bootblock_from_disk(device->fd, bblock);
705         if (ret == BC_ERROR) {
706                 (void) fprintf(stderr, gettext("Error reading bootblock from "
707                     "%s\n"), device_path);
708                 goto out_dev;
709         }

711         if (ret == BC_NOEXTRA) {
712                 BOOT_DEBUG("No multiboot header found on %s, unable "
713                     "to locate extra information area (old/non versioned "
714                     "bootblock?) \n", device_path);
715                 (void) fprintf(stderr, gettext("No extended information "
```



```
716                     "found\n"));
717                 retval = BC_NOEINFO;
718                 goto out_dev;
719         }

721         einfo = find_einfo(bblock->extra, bblock->extra_size);
719         einfo = find_einfo(bblock->extra);
722         if (einfo == NULL) {
723                 retval = BC_NOEINFO;
724                 (void) fprintf(stderr, gettext("No extended information "
725                     "found\n"));
726                 goto out_dev;
727         }

729         /* Print the extended information. */
730         if (strip)
731                 flags |= EINFO_EASY_PARSE;
732         if (verbose_dump)
733                 flags |= EINFO_PRINT_HEADER;

735         size = bblock->buf_size - P2ROUNDUP(bblock->file_size, 8) -
736             sizeof (multiboot_header_t);
737         print_einfo(flags, einfo, size);
738         retval = BC_SUCCESS;

740 out_dev:
741         cleanup_device(&data.device);
742 out:
743         free(device_path);
744         return (retval);

746 }

748 /*
749  * Attempt to mirror (propagate) the current bootblock over the attaching disk.
750  *
751  * Returns:
752  *      - BC_SUCCESS (a successful propagation happened)
753  *      - BC_ERROR (an error occurred)
754  *      - BC_NOEXTRA (it is not possible to dump the current bootblock since
755  *                    there is no multiboot information)
756  */
757 static int
758 handle_mirror(char *progname, char **argv)
759 {
760         ib_data_t       curr_data;
761         ib_data_t       attach_data;
762         ib_device_t     *curr_device = &curr_data.device;
763         ib_device_t     *attach_device = &attach_data.device;
764         ib_bootblock_t  *bblock_curr = &curr_data.bootblock;
765         ib_bootblock_t  *bblock_attach = &attach_data.bootblock;
766         bblk_einfo_t    *einfo_curr = NULL;
767         char            *curr_device_path;
768         char            *attach_device_path;
769         char            *updt_str = NULL;
770         int             retval = BC_ERROR;
771         int             ret;

773         curr_device_path = strdup(argv[0]);
774         attach_device_path = strdup(argv[1]);

776         if (!curr_device_path || !attach_device_path) {
777                 (void) fprintf(stderr, gettext("Missing parameter"));
778                 usage(progname);
779                 goto out;
780         }
```

```
781         BOOT_DEBUG("Current device path is: %s, attaching device path is: "
782             " %s\n", curr_device_path, attach_device_path);

784         bzero(&curr_data, sizeof (ib_data_t));
785         bzero(&attach_data, sizeof (ib_data_t));

787         if (tgt_fs_type != TARGET_IS_ZFS) {
788                 (void) fprintf(stderr, gettext("Mirroring is only supported on "
789                     "ZFS\n"));
790                 return (BC_ERROR);
791         }

793         if (init_device(curr_device, curr_device_path) != BC_SUCCESS) {
794                 (void) fprintf(stderr, gettext("Unable to gather device "
795                     "information from %s (current device)\n"),
796                     curr_device_path);
797                 goto out_currdev;
798         }

800         if (init_device(attach_device, attach_device_path) != BC_SUCCESS) {
801                 (void) fprintf(stderr, gettext("Unable to gather device "
802                     "information from %s (attaching device)\n"),
803                     attach_device_path);
804                 goto out_devs;
805         }

807         ret = read_bootblock_from_disk(curr_device->fd, bblock_curr);
808         if (ret == BC_ERROR) {
809                 BOOT_DEBUG("Error reading bootblock from %s\n",
810                     curr_device->path);
811                 retval = BC_ERROR;
812                 goto out_devs;
813         }

815         if (ret == BC_NOEXTRA) {
816                 BOOT_DEBUG("No multiboot header found on %s, unable to retrieve"
817                     " the bootblock\n", curr_device->path);
818                 retval = BC_NOEXTRA;
819                 goto out_devs;
820         }

822         einfo_curr = find_einfo(bblock_curr->extra, bblock_curr->extra_size);
820         einfo_curr = find_einfo(bblock_curr->extra);
823         if (einfo_curr != NULL)
824                 updt_str = einfo_get_string(einfo_curr);

826         retval = propagate_bootblock(&curr_data, &attach_data, updt_str);
827         cleanup_bootblock(bblock_curr);
828         cleanup_bootblock(bblock_attach);
829 out_devs:
830         cleanup_device(attach_device);
831 out_currdev:
832         cleanup_device(curr_device);
833 out:
834         free(curr_device_path);
835         free(attach_device_path);
836         return (retval);
837 }
```

_____***unchanged_portion_omitted_***

****
1694 Mon Jul 23 18:44:46 2012
new/usr/src/cmd/boot/installboot/installboot.h
3027 installgrub can segfault when encountering bogus data on disk
****
_____***unchanged_portion_omitted_***

```
 47 typedef struct _ib_bootblock {
 48         char                    *buf;
 49         char                    *file;
 50         char                    *extra;
 51         multiboot_header_t      *mboot;
 52         uint32_t                mboot_off;
 53         uint32_t                buf_size;
 54         uint32_t                file_size;
 55         uint32_t                extra_size;
 56 } ib_bootblock_t;
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   40477 Mon Jul 23 18:44:47 2012
new/usr/src/cmd/boot/installgrub/installgrub.c
3027 installgrub can segfault when encountering bogus data on disk
**********************************************************
______unchanged_portion_omitted_

 347 /*
 348  * Retrieves from a device the extended information (einfo) associated to the
 349  * installed stage2.
 350  * Expects one parameter, the device path, in the form: /dev/rdsk/c?[t?]d?s0.
 351  * Returns:
 352  *        - BC_SUCCESS (and prints out einfo contents depending on 'flags')
 353  *        - BC_ERROR (on error)
 354  *        - BC_NOEINFO (no extended information available)
 355  */
 356 static int
 357 handle_getinfo(char *progname, char **argv)
 358 {
 359         ig_data_t       data;
 360         ig_stage2_t     *stage2 = &data.stage2;
 361         ig_device_t     *device = &data.device;
 362         bblk_einfo_t    *einfo;
 363         uint8_t         flags = 0;
 364         uint32_t        size;
 365         char            *device_path;
 366         int             retval = BC_ERROR;
 367         int             ret;

 369         device_path = strdup(argv[0]);
 370         if (!device_path) {
 371                 (void) fprintf(stderr, gettext("Missing parameter"));
 372                 usage(progname);
 373                 goto out;
 374         }

 376         bzero(&data, sizeof (ig_data_t));
 377         BOOT_DEBUG("device path: %s\n", device_path);

 379         if (init_device(device, device_path) != BC_SUCCESS) {
 380                 (void) fprintf(stderr, gettext("Unable to gather device "
 381                     "information for %s\n"), device_path);
 382                 goto out_dev;
 383         }

 385         if (is_bootpar(device->type)) {
 386                 (void) fprintf(stderr, gettext("Versioning not supported on "
 387                     "PCFS\n"));
 388                 goto out_dev;
 389         }

 391         ret = read_stage2_from_disk(device->part_fd, stage2);
 392         if (ret == BC_ERROR) {
 393                 (void) fprintf(stderr, gettext("Error reading stage2 from "
 394                     "%s\n"), device_path);
 395                 goto out_dev;
 396         }

 398         if (ret == BC_NOEXTRA) {
 399                 (void) fprintf(stdout, gettext("No multiboot header found on "
 400                     "%s, unable to locate extra information area\n"),
 401                     device_path);
 402                 retval = BC_NOEINFO;
 403                 goto out_dev;
 404         }
```



```
 406         einfo = find_einfo(stage2->extra, stage2->extra_size);
 406         einfo = find_einfo(stage2->extra);
 407         if (einfo == NULL) {
 408                 retval = BC_NOEINFO;
 409                 (void) fprintf(stderr, gettext("No extended information "
 410                     "found\n"));
 411                 goto out_dev;
 412         }

 414         /* Print the extended information. */
 415         if (strip)
 416                 flags |= EINFO_EASY_PARSE;
 417         if (verbose_dump)
 418                 flags |= EINFO_PRINT_HEADER;

 420         size = stage2->buf_size - P2ROUNDUP(stage2->file_size, 8);
 421         print_einfo(flags, einfo, size);
 422         retval = BC_SUCCESS;

 424 out_dev:
 425         cleanup_device(&data.device);
 426 out:
 427         free(device_path);
 428         return (retval);
 429 }

 431 /*
 432  * Attempt to mirror (propagate) the current stage2 over the attaching disk.
 433  *
 434  * Returns:
 435  *      - BC_SUCCESS (a successful propagation happened)
 436  *      - BC_ERROR (an error occurred)
 437  *      - BC_NOEXTRA (it is not possible to dump the current bootblock since
 438  *                    there is no multiboot information)
 439  */
 440 static int
 441 handle_mirror(char *progname, char **argv)
 442 {
 443         ig_data_t       curr_data;
 444         ig_data_t       attach_data;
 445         ig_device_t     *curr_device = &curr_data.device;
 446         ig_device_t     *attach_device = &attach_data.device;
 447         ig_stage2_t     *stage2_curr = &curr_data.stage2;
 448         ig_stage2_t     *stage2_attach = &attach_data.stage2;
 449         bblk_einfo_t    *einfo_curr = NULL;
 450         char            *curr_device_path;
 451         char            *attach_device_path;
 452         char            *updt_str = NULL;
 453         int             retval = BC_ERROR;
 454         int             ret;

 456         curr_device_path = strdup(argv[0]);
 457         attach_device_path = strdup(argv[1]);

 459         if (!curr_device_path || !attach_device_path) {
 460                 (void) fprintf(stderr, gettext("Missing parameter"));
 461                 usage(progname);
 462                 goto out;
 463         }
 464         BOOT_DEBUG("Current device path is: %s, attaching device path is: "
 465             " %s\n", curr_device_path, attach_device_path);

 467         bzero(&curr_data, sizeof (ig_data_t));
 468         bzero(&attach_data, sizeof (ig_data_t));

 470         if (init_device(curr_device, curr_device_path) != BC_SUCCESS) {
```

```
 471                 (void) fprintf(stderr, gettext("Unable to gather device "
 472                     "information for %s (current device)\n"), curr_device_path);
 473                 goto out_currdev;
 474         }

 476         if (init_device(attach_device, attach_device_path) != BC_SUCCESS) {
 477                 (void) fprintf(stderr, gettext("Unable to gather device "
 478                     "information for %s (attaching device)\n"),
 479                     attach_device_path);
 480                 goto out_devs;
 481         }

 483         if (is_bootpar(curr_device->type) || is_bootpar(attach_device->type)) {
 484                 (void) fprintf(stderr, gettext("boot block mirroring is not "
 485                     "supported on PCFS\n"));
 486                 goto out_devs;
 487         }

 489         ret = read_stage2_from_disk(curr_device->part_fd, stage2_curr);
 490         if (ret == BC_ERROR) {
 491                 BOOT_DEBUG("Error reading first stage2 blocks from %s\n",
 492                     curr_device->path);
 493                 retval = BC_ERROR;
 494                 goto out_devs;
 495         }

 497         if (ret == BC_NOEXTRA) {
 498                 BOOT_DEBUG("No multiboot header found on %s, unable to grab "
 499                     "stage2\n", curr_device->path);
 500                 retval = BC_NOEXTRA;
 501                 goto out_devs;
 502         }

 504         einfo_curr = find_einfo(stage2_curr->extra, stage2_curr->extra_size);
 504         einfo_curr = find_einfo(stage2_curr->extra);
 505         if (einfo_curr != NULL)
 506                 updt_str = einfo_get_string(einfo_curr);

 508         write_mbr = B_TRUE;
 509         force_mbr = B_TRUE;
 510         retval = propagate_bootblock(&curr_data, &attach_data, updt_str);
 511         cleanup_stage2(stage2_curr);
 512         cleanup_stage2(stage2_attach);

 514 out_devs:
 515         cleanup_device(attach_device);
 516 out_currdev:
 517         cleanup_device(curr_device);
 518 out:
 519         free(curr_device_path);
 520         free(attach_device_path);
 521         return (retval);
 522 }
```

unchanged_portion_omitted_

```
1163 static int
1164 read_stage2_from_disk(int dev_fd, ig_stage2_t *stage2)
1165 {
1166         uint32_t                size;
1167         uint32_t                buf_size;
1168         uint32_t                mboot_off;
1169         multiboot_header_t      *mboot;

1171         assert(stage2 != NULL);
1172         assert(dev_fd != -1);
```



```
1174         if (read_in(dev_fd, mboot_scan, sizeof (mboot_scan),
1175             STAGE2_BLKOFF * SECTOR_SIZE) != BC_SUCCESS) {
1176                 perror(gettext("Error reading stage2 sectors"));
1177                 return (BC_ERROR);
1178         }

1180         /* No multiboot means no chance of knowing stage2 size */
1181         if (find_multiboot(mboot_scan, sizeof (mboot_scan), &mboot_off)
1182             != BC_SUCCESS) {
1183                 BOOT_DEBUG("Unable to find multiboot header\n");
1184                 return (BC_NOEXTRA);
1185         }
1186         mboot = (multiboot_header_t *)(mboot_scan + mboot_off);

1188         /*
1189          * Unfilled mboot values mean an older version of installgrub installed
1190          * the stage2. Again we have no chance of knowing stage2 size.
1191          */
1192         if (mboot->load_end_addr == 0 ||
1193             mboot->load_end_addr < mboot->load_addr)
1194                 return (BC_NOEXTRA);

1196         /*
1197          * Currently, the amount of space reserved for extra information
1198          * is "fixed". We may have to scan for the terminating extra payload
1199          * in the future.
1200          */
1201         size = mboot->load_end_addr - mboot->load_addr;
1202         buf_size = P2ROUNDUP(size + SECTOR_SIZE, SECTOR_SIZE);

1204         stage2->buf = malloc(buf_size);
1205         if (stage2->buf == NULL) {
1206                 perror(gettext("Memory allocation failed"));
1207                 return (BC_ERROR);
1208         }
1209         stage2->buf_size = buf_size;

1211         if (read_in(dev_fd, stage2->buf, buf_size, STAGE2_BLKOFF *
1212             SECTOR_SIZE) != BC_SUCCESS) {
1213                 perror("read");
1214                 free(stage2->buf);
1215                 return (BC_ERROR);
1216         }

1218         /* Update pointers. */
1219         stage2->file = stage2->buf;
1220         stage2->file_size = size;
1221         stage2->mboot_off = mboot_off;
1222         stage2->mboot = (multiboot_header_t *)(stage2->buf + stage2->mboot_off);
1223         stage2->extra = stage2->buf + P2ROUNDUP(stage2->file_size, 8);
1224         stage2->extra_size = stage2->buf_size - P2ROUNDUP(stage2->file_size, 8);

1226         return (BC_SUCCESS);
1227 }

1229 static boolean_t
1230 is_update_necessary(ig_data_t *data, char *updt_str)
1231 {
1232         bblk_einfo_t    *einfo;
1233         bblk_hs_t       stage2_hs;
1234         ig_stage2_t     stage2_disk;
1235         ig_stage2_t     *stage2_file = &data->stage2;
1236         ig_device_t     *device = &data->device;
1237         int             dev_fd = device->part_fd;

1239         assert(data != NULL);
```

```
1240         assert(device->part_fd != -1);

1242         bzero(&stage2_disk, sizeof (ig_stage2_t));

1244         /* Gather stage2 (if present) from the target device. */
1245         if (read_stage2_from_disk(dev_fd, &stage2_disk) != BC_SUCCESS) {
1246                 BOOT_DEBUG("Unable to read stage2 from %s\n", device->path);
1247                 BOOT_DEBUG("No multiboot wrapped stage2 on %s\n", device->path);
1248                 return (B_TRUE);
1249         }

1251         /*
1252          * Look for the extended information structure in the extra payload
1253          * area.
1254          */
1255         einfo = find_einfo(stage2_disk.extra, stage2_disk.extra_size);
1254         einfo = find_einfo(stage2_disk.extra);
1256         if (einfo == NULL) {
1257                 BOOT_DEBUG("No extended information available\n");
1258                 return (B_TRUE);
1259         }

1261         if (!do_version || updt_str == NULL) {
1262                 (void) fprintf(stdout, "WARNING: target device %s has a "
1263                     "versioned stage2 that is going to be overwritten by a non "
1264                     "versioned one\n", device->path);
1265                 return (B_TRUE);
1266         }

1268         if (force_update) {
1269                 BOOT_DEBUG("Forcing update of %s bootblock\n", device->path);
1270                 return (B_TRUE);
1271         }

1273         /* Compare the two extended information structures. */
1274         stage2_hs.src_buf = (unsigned char *)stage2_file->file;
1275         stage2_hs.src_size = stage2_file->file_size;

1277         return (einfo_should_update(einfo, &stage2_hs, updt_str));
1278 }
```

_____***unchanged_portion_omitted_***

****
2353 Mon Jul 23 18:44:48 2012
new/usr/src/cmd/boot/installgrub/installgrub.h
3027 installgrub can segfault when encountering bogus data on disk
****
_____***unchanged_portion_omitted_***

```
 49 typedef struct _stage2_data {
 50         char                    *buf;
 51         char                    *file;
 52         char                    *extra;
 53         multiboot_header_t      *mboot;
 54         uint32_t                mboot_off;
 55         uint32_t                file_size;
 56         uint32_t                extra_size;
 57         uint32_t                buf_size;
 58         uint32_t                first_sector;
 59         uint32_t                pcfs_first_sectors[2];
 60 } ig_stage2_t;
```

_____***unchanged_portion_omitted_***