```
**********************************************************
   28711 Tue Feb 11 13:26:06 2014
new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_dhcpv6.c
4587 snoop misdecodes DHCPv6 DHCPV6_DUID_LL identifiers
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Dynamic Host Configuration Protocol version 6, for IPv6.  Supports
 * RFCs 3315, 3319, 3646, 3898, 4075, 4242, 4280, 4580, 4649, and 4704.
 */

#include <ctype.h>
#endif /* ! codereview */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/dhcp6.h>
#include <arpa/inet.h>
#include <dhcp_impl.h>
#include <dhcp_inittab.h>

#include "snoop.h"

static const char *mtype_to_str(uint8_t);
static const char *option_to_str(uint8_t);
static const char *duidtype_to_str(uint16_t);
static const char *status_to_str(uint16_t);
static const char *entr_to_str(uint32_t);
static const char *reconf_to_str(uint8_t);
static const char *authproto_to_str(uint8_t);
static const char *authalg_to_str(uint8_t, uint8_t);
static const char *authrdm_to_str(uint8_t);
static const char *cwhat_to_str(uint8_t);
static const char *catype_to_str(uint8_t);
static void show_hex(const uint8_t *, int, const char *);
static void show_ascii(const uint8_t *, int, const char *);
```



```
static void show_address(const char *, const void *);
static void show_options(const uint8_t *, int);

int
interpret_dhcpv6(int flags, const uint8_t *data, int len)
{
        int olen = len;
        char *line, *lstart;
        dhcpv6_relay_t d6r;
        dhcpv6_message_t d6m;
        uint_t optlen;
        uint16_t statuscode;

        if (len <= 0) {
                (void) strlcpy(get_sum_line(), "DHCPv6?", MAXLINE);
                return (0);
        }
        if (flags & F_SUM) {
                uint_t ias;
                dhcpv6_option_t *d6o;
                in6_addr_t link, peer;
                char linkstr[INET6_ADDRSTRLEN];
                char peerstr[INET6_ADDRSTRLEN];

                line = lstart = get_sum_line();
                line += snprintf(line, MAXLINE, "DHCPv6 %s",
                    mtype_to_str(data[0]));
                if (data[0] == DHCPV6_MSG_RELAY_FORW ||
                    data[0] == DHCPV6_MSG_RELAY_REPL) {
                        if (len < sizeof (d6r)) {
                                (void) strlcpy(line, "?",
                                    MAXLINE - (line - lstart));
                                return (olen);
                        }
                        /* Not much in DHCPv6 is aligned. */
                        (void) memcpy(&d6r, data, sizeof (d6r));
                        (void) memcpy(&link, d6r.d6r_linkaddr, sizeof (link));
                        (void) memcpy(&peer, d6r.d6r_peeraddr, sizeof (peer));
                        line += snprintf(line, MAXLINE - (line - lstart),
                            " HC=%d link=%s peer=%s", d6r.d6r_hop_count,
                            inet_ntop(AF_INET6, &link, linkstr,
                            sizeof (linkstr)),
                            inet_ntop(AF_INET6, &peer, peerstr,
                            sizeof (peerstr)));
                        data += sizeof (d6r);
                        len -= sizeof (d6r);
                } else {
                        if (len < sizeof (d6m)) {
                                (void) strlcpy(line, "?",
                                    MAXLINE - (line - lstart));
                                return (olen);
                        }
                        (void) memcpy(&d6m, data, sizeof (d6m));
                        line += snprintf(line, MAXLINE - (line - lstart),
                            " xid=%x", DHCPV6_GET_TRANSID(&d6m));
                        data += sizeof (d6m);
                        len -= sizeof (d6m);
                }
                ias = 0;
                d6o = NULL;
                while ((d6o = dhcpv6_find_option(data, len, d6o,
                    DHCPV6_OPT_IA_NA, NULL)) != NULL)
                        ias++;
                if (ias > 0)
                        line += snprintf(line, MAXLINE - (line - lstart),
                            " IAs=%u", ias);
```

```
127                 d6o = dhcpv6_find_option(data, len, NULL,
128                     DHCPV6_OPT_STATUS_CODE, &optlen);
129                 optlen -= sizeof (*d6o);
130                 if (d6o != NULL && optlen >= sizeof (statuscode)) {
131                         (void) memcpy(&statuscode, d6o + 1,
132                             sizeof (statuscode));
133                         line += snprintf(line, MAXLINE - (line - lstart),
134                             " status=%u", ntohs(statuscode));
135                         optlen -= sizeof (statuscode);
136                         if (optlen > 0) {
137                                 line += snprintf(line,
138                                     MAXLINE - (line - lstart), " \"%.*s\"",
139                                     optlen, (char *)(d6o + 1) + 2);
140                         }
141                 }
142                 d6o = dhcpv6_find_option(data, len, NULL,
143                     DHCPV6_OPT_RELAY_MSG, &optlen);
144                 optlen -= sizeof (*d6o);
145                 if (d6o != NULL && optlen >= 1) {
146                         line += snprintf(line, MAXLINE - (line - lstart),
147                             " relay=%s", mtype_to_str(*(uint8_t *)(d6o + 1)));
148                 }
149         } else if (flags & F_DTAIL) {
150                 show_header("DHCPv6: ",
151                     "Dynamic Host Configuration Protocol Version 6", len);
152                 show_space();
153                 (void) snprintf(get_line(0, 0), get_line_remain(),
154                     "Message type (msg-type) = %u (%s)", data[0],
155                     mtype_to_str(data[0]));
156                 if (data[0] == DHCPV6_MSG_RELAY_FORW ||
157                     data[0] == DHCPV6_MSG_RELAY_REPL) {
158                         if (len < sizeof (d6r)) {
159                                 (void) strlcpy(get_line(0, 0), "Truncated",
160                                     get_line_remain());
161                                 return (olen);
162                         }
163                         (void) memcpy(&d6r, data, sizeof (d6r));
164                         (void) snprintf(get_line(0, 0), get_line_remain(),
165                             "Hop count = %u", d6r.d6r_hop_count);
166                         show_address("Link address", d6r.d6r_linkaddr);
167                         show_address("Peer address", d6r.d6r_peeraddr);
168                         data += sizeof (d6r);
169                         len -= sizeof (d6r);
170                 } else {
171                         if (len < sizeof (d6m)) {
172                                 (void) strlcpy(get_line(0, 0), "Truncated",
173                                     get_line_remain());
174                                 return (olen);
175                         }
176                         (void) memcpy(&d6m, data, sizeof (d6m));
177                         (void) snprintf(get_line(0, 0), get_line_remain(),
178                             "Transaction ID = %x", DHCPV6_GET_TRANSID(&d6m));
179                         data += sizeof (d6m);
180                         len -= sizeof (d6m);
181                 }
182                 show_space();
183                 show_options(data, len);
184                 show_space();
185         }
186         return (olen);
187 }

189 static const char *
190 mtype_to_str(uint8_t mtype)
191 {
192         switch (mtype) {
```



```
193         case DHCPV6_MSG_SOLICIT:
194                 return ("Solicit");
195         case DHCPV6_MSG_ADVERTISE:
196                 return ("Advertise");
197         case DHCPV6_MSG_REQUEST:
198                 return ("Request");
199         case DHCPV6_MSG_CONFIRM:
200                 return ("Confirm");
201         case DHCPV6_MSG_RENEW:
202                 return ("Renew");
203         case DHCPV6_MSG_REBIND:
204                 return ("Rebind");
205         case DHCPV6_MSG_REPLY:
206                 return ("Reply");
207         case DHCPV6_MSG_RELEASE:
208                 return ("Release");
209         case DHCPV6_MSG_DECLINE:
210                 return ("Decline");
211         case DHCPV6_MSG_RECONFIGURE:
212                 return ("Reconfigure");
213         case DHCPV6_MSG_INFO_REQ:
214                 return ("Information-Request");
215         case DHCPV6_MSG_RELAY_FORW:
216                 return ("Relay-Forward");
217         case DHCPV6_MSG_RELAY_REPL:
218                 return ("Relay-Reply");
219         default:
220                 return ("Unknown");
221         }
222 }

224 static const char *
225 option_to_str(uint8_t mtype)
226 {
227         switch (mtype) {
228         case DHCPV6_OPT_CLIENTID:
229                 return ("Client Identifier");
230         case DHCPV6_OPT_SERVERID:
231                 return ("Server Identifier");
232         case DHCPV6_OPT_IA_NA:
233                 return ("Identity Association for Non-temporary Addresses");
234         case DHCPV6_OPT_IA_TA:
235                 return ("Identity Association for Temporary Addresses");
236         case DHCPV6_OPT_IAADDR:
237                 return ("IA Address");
238         case DHCPV6_OPT_ORO:
239                 return ("Option Request");
240         case DHCPV6_OPT_PREFERENCE:
241                 return ("Preference");
242         case DHCPV6_OPT_ELAPSED_TIME:
243                 return ("Elapsed Time");
244         case DHCPV6_OPT_RELAY_MSG:
245                 return ("Relay Message");
246         case DHCPV6_OPT_AUTH:
247                 return ("Authentication");
248         case DHCPV6_OPT_UNICAST:
249                 return ("Server Unicast");
250         case DHCPV6_OPT_STATUS_CODE:
251                 return ("Status Code");
252         case DHCPV6_OPT_RAPID_COMMIT:
253                 return ("Rapid Commit");
254         case DHCPV6_OPT_USER_CLASS:
255                 return ("User Class");
256         case DHCPV6_OPT_VENDOR_CLASS:
257                 return ("Vendor Class");
258         case DHCPV6_OPT_VENDOR_OPT:
```

```
259                 return ("Vendor-specific Information");
260         case DHCPV6_OPT_INTERFACE_ID:
261                 return ("Interface-Id");
262         case DHCPV6_OPT_RECONF_MSG:
263                 return ("Reconfigure Message");
264         case DHCPV6_OPT_RECONF_ACC:
265                 return ("Reconfigure Accept");
266         case DHCPV6_OPT_SIP_NAMES:
267                 return ("SIP Servers Domain Name List");
268         case DHCPV6_OPT_SIP_ADDR:
269                 return ("SIP Servers IPv6 Address List");
270         case DHCPV6_OPT_DNS_ADDR:
271                 return ("DNS Recursive Name Server");
272         case DHCPV6_OPT_DNS_SEARCH:
273                 return ("Domain Search List");
274         case DHCPV6_OPT_IA_PD:
275                 return ("Identity Association for Prefix Delegation");
276         case DHCPV6_OPT_IAPREFIX:
277                 return ("IA_PD Prefix");
278         case DHCPV6_OPT_NIS_SERVERS:
279                 return ("Network Information Service Servers");
280         case DHCPV6_OPT_NIS_DOMAIN:
281                 return ("Network Information Service Domain Name");
282         case DHCPV6_OPT_SNTP_SERVERS:
283                 return ("Simple Network Time Protocol Servers");
284         case DHCPV6_OPT_INFO_REFTIME:
285                 return ("Information Refresh Time");
286         case DHCPV6_OPT_BCMCS_SRV_D:
287                 return ("BCMCS Controller Domain Name List");
288         case DHCPV6_OPT_BCMCS_SRV_A:
289                 return ("BCMCS Controller IPv6 Address");
290         case DHCPV6_OPT_GEOCONF_CVC:
291                 return ("Civic Location");
292         case DHCPV6_OPT_REMOTE_ID:
293                 return ("Relay Agent Remote-ID");
294         case DHCPV6_OPT_SUBSCRIBER:
295                 return ("Relay Agent Subscriber-ID");
296         case DHCPV6_OPT_CLIENT_FQDN:
297                 return ("Client FQDN");
298         default:
299                 return ("Unknown");
300         }
301 }

303 static const char *
304 duidtype_to_str(uint16_t dtype)
305 {
306         switch (dtype) {
307         case DHCPV6_DUID_LLT:
308                 return ("Link-layer Address Plus Time");
309         case DHCPV6_DUID_EN:
310                 return ("Enterprise Number");
311         case DHCPV6_DUID_LL:
312                 return ("Link-layer Address");
313         default:
314                 return ("Unknown");
315         }
316 }

318 static const char *
319 status_to_str(uint16_t status)
320 {
321         switch (status) {
322         case DHCPV6_STAT_SUCCESS:
323                 return ("Success");
324         case DHCPV6_STAT_UNSPECFAIL:
```



```
325                 return ("Failure, reason unspecified");
326         case DHCPV6_STAT_NOADDRS:
327                 return ("No addresses for IAs");
328         case DHCPV6_STAT_NOBINDING:
329                 return ("Client binding unavailable");
330         case DHCPV6_STAT_NOTONLINK:
331                 return ("Prefix not on link");
332         case DHCPV6_STAT_USEMCAST:
333                 return ("Use multicast");
334         case DHCPV6_STAT_NOPREFIX:
335                 return ("No prefix available");
336         default:
337                 return ("Unknown");
338         }
339 }

341 static const char *
342 entr_to_str(uint32_t entr)
343 {
344         switch (entr) {
345         case DHCPV6_SUN_ENT:
346                 return ("Sun Microsystems");
347         default:
348                 return ("Unknown");
349         }
350 }

352 static const char *
353 reconf_to_str(uint8_t msgtype)
354 {
355         switch (msgtype) {
356         case DHCPV6_RECONF_RENEW:
357                 return ("Renew");
358         case DHCPV6_RECONF_INFO:
359                 return ("Information-request");
360         default:
361                 return ("Unknown");
362         }
363 }

365 static const char *
366 authproto_to_str(uint8_t aproto)
367 {
368         switch (aproto) {
369         case DHCPV6_PROTO_DELAYED:
370                 return ("Delayed");
371         case DHCPV6_PROTO_RECONFIG:
372                 return ("Reconfigure Key");
373         default:
374                 return ("Unknown");
375         }
376 }

378 static const char *
379 authalg_to_str(uint8_t aproto, uint8_t aalg)
380 {
381         switch (aproto) {
382         case DHCPV6_PROTO_DELAYED:
383         case DHCPV6_PROTO_RECONFIG:
384                 switch (aalg) {
385                 case DHCPV6_ALG_HMAC_MD5:
386                         return ("HMAC-MD5 Signature");
387                 default:
388                         return ("Unknown");
389                 }
390                 break;
```

```
391         default:
392                 return ("Unknown");
393         }
394 }

396 static const char *
397 authrdm_to_str(uint8_t ardm)
398 {
399         switch (ardm) {
400         case DHCPV6_RDM_MONOCNT:
401                 return ("Monotonic Counter");
402         default:
403                 return ("Unknown");
404         }
405 }

407 static const char *
408 cwhat_to_str(uint8_t what)
409 {
410         switch (what) {
411         case DHCPV6_CWHAT_SERVER:
412                 return ("Server");
413         case DHCPV6_CWHAT_NETWORK:
414                 return ("Network");
415         case DHCPV6_CWHAT_CLIENT:
416                 return ("Client");
417         default:
418                 return ("Unknown");
419         }
420 }

422 static const char *
423 catype_to_str(uint8_t catype)
424 {
425         switch (catype) {
426         case CIVICADDR_LANG:
427                 return ("Language; RFC 2277");
428         case CIVICADDR_A1:
429                 return ("National division (state)");
430         case CIVICADDR_A2:
431                 return ("County");
432         case CIVICADDR_A3:
433                 return ("City");
434         case CIVICADDR_A4:
435                 return ("City division");
436         case CIVICADDR_A5:
437                 return ("Neighborhood");
438         case CIVICADDR_A6:
439                 return ("Street group");
440         case CIVICADDR_PRD:
441                 return ("Leading street direction");
442         case CIVICADDR_POD:
443                 return ("Trailing street suffix");
444         case CIVICADDR_STS:
445                 return ("Street suffix or type");
446         case CIVICADDR_HNO:
447                 return ("House number");
448         case CIVICADDR_HNS:
449                 return ("House number suffix");
450         case CIVICADDR_LMK:
451                 return ("Landmark");
452         case CIVICADDR_LOC:
453                 return ("Additional location information");
454         case CIVICADDR_NAM:
455                 return ("Name/occupant");
456         case CIVICADDR_PC:
```



```
457                 return ("Postal Code/ZIP");
458         case CIVICADDR_BLD:
459                 return ("Building");
460         case CIVICADDR_UNIT:
461                 return ("Unit/apt/suite");
462         case CIVICADDR_FLR:
463                 return ("Floor");
464         case CIVICADDR_ROOM:
465                 return ("Room number");
466         case CIVICADDR_TYPE:
467                 return ("Place type");
468         case CIVICADDR_PCN:
469                 return ("Postal community name");
470         case CIVICADDR_POBOX:
471                 return ("Post office box");
472         case CIVICADDR_ADDL:
473                 return ("Additional code");
474         case CIVICADDR_SEAT:
475                 return ("Seat/desk");
476         case CIVICADDR_ROAD:
477                 return ("Primary road or street");
478         case CIVICADDR_RSEC:
479                 return ("Road section");
480         case CIVICADDR_RBRA:
481                 return ("Road branch");
482         case CIVICADDR_RSBR:
483                 return ("Road sub-branch");
484         case CIVICADDR_SPRE:
485                 return ("Street name pre-modifier");
486         case CIVICADDR_SPOST:
487                 return ("Street name post-modifier");
488         case CIVICADDR_SCRIPT:
489                 return ("Script");
490         default:
491                 return ("Unknown");
492         }
493 }

495 static void
496 show_hex(const uint8_t *data, int len, const char *name)
497 {
498         char buffer[16 * 3 + 1];
499         int nlen;
500         int i;
501         char sep;

503         nlen = strlen(name);
504         sep = '=';
505         while (len > 0) {
506                 for (i = 0; i < 16 && i < len; i++)
507                         (void) snprintf(buffer + 3 * i, 4, " %02x", *data++);
508                 (void) snprintf(get_line(0, 0), get_line_remain(), "%*s %c%s",
509                     nlen, name, sep, buffer);
510                 name = "";
511                 sep = ' ';
512                 len -= i;
513         }
514 }

516 static void
517 show_ascii(const uint8_t *data, int len, const char *name)
518 {
519         char buffer[64], *bp;
520         int nlen;
521         int i;
522         char sep;
```

```
524         nlen = strlen(name);
525         sep = '=';
526         while (len > 0) {
527                 bp = buffer;
528                 for (i = 0; i < sizeof (buffer) - 4 && len > 0; len--) {
529                         if (!isascii(*data) || !isprint(*data))
530                                 bp += snprintf(bp, 5, "\\%03o", *data++);
531                         else
532                                 *bp++;
533                 }
534                 *bp = '\0';
535                 (void) snprintf(get_line(0, 0), get_line_remain(),
536                     "%*s %c \"%s\"", nlen, name, sep, buffer);
537                 sep = ' ';
538                 name = "";
539         }
540 }

542 static void
543 show_address(const char *addrname, const void *aptr)
544 {
545         char *hname;
546         char addrstr[INET6_ADDRSTRLEN];
547         in6_addr_t addr;

549         (void) memcpy(&addr, aptr, sizeof (in6_addr_t));
550         (void) inet_ntop(AF_INET6, &addr, addrstr, sizeof (addrstr));
551         hname = addrtoname(AF_INET6, &addr);
552         if (strcmp(hname, addrstr) == 0) {
553                 (void) snprintf(get_line(0, 0), get_line_remain(), "%s = %s",
554                     addrname, addrstr);
555         } else {
556                 (void) snprintf(get_line(0, 0), get_line_remain(),
557                     "%s = %s (%s)", addrname, addrstr, hname);
558         }
559 }

561 static void
562 nest_options(const uint8_t *data, uint_t olen, char *prefix, char *title)
563 {
564         char *str, *oldnest, *oldprefix;

566         if (olen <= 0)
567                 return;
568         oldprefix = prot_prefix;
569         oldnest = prot_nest_prefix;
570         str = malloc(strlen(prot_nest_prefix) + strlen(prot_prefix) + 1);
571         if (str == NULL) {
572                 prot_nest_prefix = prot_prefix;
573         } else {
574                 (void) sprintf(str, "%s%s", prot_nest_prefix, prot_prefix);
575                 prot_nest_prefix = str;
576         }
577         show_header(prefix, title, 0);
578         show_options(data, olen);
579         free(str);
580         prot_prefix = oldprefix;
581         prot_nest_prefix = oldnest;
582 }

584 static void
585 show_options(const uint8_t *data, int len)
586 {
587         dhcpv6_option_t d6o;
588         uint_t olen, retlen;
```



```
589         uint16_t val16;
590         uint16_t type;
591         uint32_t val32;
592         const uint8_t *ostart;
593         char *str, *sp;
594         char *oldnest;

596         /*
597          * Be very careful with negative numbers; ANSI signed/unsigned
598          * comparison doesn't work as expected.
599          */
600         while (len >= (signed)sizeof (d6o)) {
601                 (void) memcpy(&d6o, data, sizeof (d6o));
602                 d6o.d6o_code = ntohs(d6o.d6o_code);
603                 d6o.d6o_len = olen = ntohs(d6o.d6o_len);
604                 (void) snprintf(get_line(0, 0), get_line_remain(),
605                     "Option Code = %u (%s)", d6o.d6o_code,
606                     option_to_str(d6o.d6o_code));
607                 ostart = data += sizeof (d6o);
608                 len -= sizeof (d6o);
609                 if (olen > len) {
610                         (void) strlcpy(get_line(0, 0), "Option truncated",
611                             get_line_remain());
612                         olen = len;
613                 }
614                 switch (d6o.d6o_code) {
615                 case DHCPV6_OPT_CLIENTID:
616                 case DHCPV6_OPT_SERVERID:
617                         if (olen < sizeof (val16))
618                                 break;
619                         (void) memcpy(&val16, data, sizeof (val16));
620                         data += sizeof (val16);
621                         olen -= sizeof (val16);
622                         type = ntohs(val16);
623                         (void) snprintf(get_line(0, 0), get_line_remain(),
624                             "  DUID Type = %u (%s)", type,
625                             duidtype_to_str(type));
626                         if (type == DHCPV6_DUID_LLT || type == DHCPV6_DUID_LL) {
627                                 if (olen < sizeof (val16))
628                                         break;
629                                 (void) memcpy(&val16, data, sizeof (val16));
630                                 data += sizeof (val16);
631                                 olen -= sizeof (val16);
632                                 val16 = ntohs(val16);
633                                 (void) snprintf(get_line(0, 0),
634                                     get_line_remain(),
635                                     "  Hardware Type = %u (%s)", val16,
636                                     arp_htype(val16));
 32                                  arp_htype(type));
637                         }
638                         if (type == DHCPV6_DUID_LLT) {
639                                 time_t timevalue;

641                                 if (olen < sizeof (val32))
642                                         break;
643                                 (void) memcpy(&val32, data, sizeof (val32));
644                                 data += sizeof (val32);
645                                 olen -= sizeof (val32);
646                                 timevalue = ntohl(val32) + DUID_TIME_BASE;
647                                 (void) snprintf(get_line(0, 0),
648                                     get_line_remain(),
649                                     "  Time = %lu (%.24s)", ntohl(val32),
650                                     ctime(&timevalue));
651                         }
652                         if (type == DHCPV6_DUID_EN) {
653                                 if (olen < sizeof (val32))
```

```
654                                     break;
655                             (void) memcpy(&val32, data, sizeof (val32));
656                             data += sizeof (val32);
657                             olen -= sizeof (val32);
658                             val32 = ntohl(val32);
659                             (void) snprintf(get_line(0, 0),
660                                 get_line_remain(),
661                                 "  Enterprise Number = %lu (%s)", val32,
662                                 entr_to_str(val32));
663                     }
664                     if (olen == 0)
665                             break;
666                     if ((str = malloc(olen * 3)) == NULL)
667                             pr_err("interpret_dhcpv6: no mem");
668                     sp = str + snprintf(str, 3, "%02x", *data++);
669                     while (--olen > 0) {
670                             *sp++ = (type == DHCPV6_DUID_LLT ||
671                                 type == DHCPV6_DUID_LL) ? ':' : ' ';
672                             sp = sp + snprintf(sp, 3, "%02x", *data++);
673                     }
674                     (void) snprintf(get_line(0, 0), get_line_remain(),
675                         (type == DHCPV6_DUID_LLT ||
676                         type == DHCPV6_DUID_LL) ?
677                         "  Link Layer Address = %s" :
678                         "  Identifier = %s", str);
679                     free(str);
680                     break;
681             case DHCPV6_OPT_IA_NA:
682             case DHCPV6_OPT_IA_PD: {
683                     dhcpv6_ia_na_t d6in;

685                     if (olen < sizeof (d6in) - sizeof (d6o))
686                             break;
687                     (void) memcpy(&d6in, data - sizeof (d6o),
688                         sizeof (d6in));
689                     data += sizeof (d6in) - sizeof (d6o);
690                     olen -= sizeof (d6in) - sizeof (d6o);
691                     (void) snprintf(get_line(0, 0), get_line_remain(),
692                         "  IAID = %u", ntohl(d6in.d6in_iaid));
693                     (void) snprintf(get_line(0, 0), get_line_remain(),
694                         "  T1 (renew) = %u seconds", ntohl(d6in.d6in_t1));
695                     (void) snprintf(get_line(0, 0), get_line_remain(),
696                         "  T2 (rebind) = %u seconds", ntohl(d6in.d6in_t2));
697                     nest_options(data, olen, "IA: ",
698                         "Identity Association");
699                     break;
700             }
701             case DHCPV6_OPT_IA_TA: {
702                     dhcpv6_ia_ta_t d6it;

704                     if (olen < sizeof (d6it) - sizeof (d6o))
705                             break;
706                     (void) memcpy(&d6it, data - sizeof (d6o),
707                         sizeof (d6it));
708                     data += sizeof (d6it) - sizeof (d6o);
709                     olen -= sizeof (d6it) - sizeof (d6o);
710                     (void) snprintf(get_line(0, 0), get_line_remain(),
711                         "  IAID = %u", ntohl(d6it.d6it_iaid));
712                     nest_options(data, olen, "IA: ",
713                         "Identity Association");
714                     break;
715             }
716             case DHCPV6_OPT_IAADDR: {
717                     dhcpv6_iaaddr_t d6ia;

719                     if (olen < sizeof (d6ia) - sizeof (d6o))
```



```
720                             break;
721                     (void) memcpy(&d6ia, data - sizeof (d6o),
722                         sizeof (d6ia));
723                     data += sizeof (d6ia) - sizeof (d6o);
724                     olen -= sizeof (d6ia) - sizeof (d6o);
725                     show_address("  Address", &d6ia.d6ia_addr);
726                     (void) snprintf(get_line(0, 0), get_line_remain(),
727                         "  Preferred lifetime = %u seconds",
728                         ntohl(d6ia.d6ia_preflife));
729                     (void) snprintf(get_line(0, 0), get_line_remain(),
730                         "  Valid lifetime = %u seconds",
731                         ntohl(d6ia.d6ia_vallife));
732                     nest_options(data, olen, "ADDR: ", "Address");
733                     break;
734             }
735             case DHCPV6_OPT_ORO:
736                     while (olen >= sizeof (val16)) {
737                             (void) memcpy(&val16, data, sizeof (val16));
738                             val16 = ntohs(val16);
739                             (void) snprintf(get_line(0, 0),
740                                 get_line_remain(),
741                                 "  Requested Option Code = %u (%s)", val16,
742                                 option_to_str(val16));
743                             data += sizeof (val16);
744                             olen -= sizeof (val16);
745                     }
746                     break;
747             case DHCPV6_OPT_PREFERENCE:
748                     if (olen > 0) {
749                             (void) snprintf(get_line(0, 0),
750                                 get_line_remain(),
751                                 *data == 255 ?
752                                 "  Preference = %u (immediate)" :
753                                 "  Preference = %u", *data);
754                     }
755                     break;
756             case DHCPV6_OPT_ELAPSED_TIME:
757                     if (olen == sizeof (val16)) {
758                             (void) memcpy(&val16, data, sizeof (val16));
759                             val16 = ntohs(val16);
760                             (void) snprintf(get_line(0, 0),
761                                 get_line_remain(),
762                                 "  Elapsed Time = %u.%02u seconds",
763                                 val16 / 100, val16 % 100);
764                     }
765                     break;
766             case DHCPV6_OPT_RELAY_MSG:
767                     if (olen > 0) {
768                             oldnest = prot_nest_prefix;
769                             prot_nest_prefix = prot_prefix;
770                             retlen = interpret_dhcpv6(F_DTAIL, data, olen);
771                             prot_prefix = prot_nest_prefix;
772                             prot_nest_prefix = oldnest;
773                     }
774                     break;
775             case DHCPV6_OPT_AUTH: {
776                     dhcpv6_auth_t d6a;

778                     if (olen < DHCPV6_AUTH_SIZE - sizeof (d6o))
779                             break;
780                     (void) memcpy(&d6a, data - sizeof (d6o),
781                         DHCPV6_AUTH_SIZE);
782                     data += DHCPV6_AUTH_SIZE - sizeof (d6o);
783                     olen += DHCPV6_AUTH_SIZE - sizeof (d6o);
784                     (void) snprintf(get_line(0, 0), get_line_remain(),
785                         "  Protocol = %u (%s)", d6a.d6a_proto,
```

```
786                         authproto_to_str(d6a.d6a_proto));
787                 (void) snprintf(get_line(0, 0), get_line_remain(),
788                     "  Algorithm = %u (%s)", d6a.d6a_alg,
789                     authalg_to_str(d6a.d6a_proto, d6a.d6a_alg));
790                 (void) snprintf(get_line(0, 0), get_line_remain(),
791                     "  Replay Detection Method = %u (%s)", d6a.d6a_rdm,
792                     authrdm_to_str(d6a.d6a_rdm));
793                 show_hex(d6a.d6a_replay, sizeof (d6a.d6a_replay),
794                     "  RDM Data");
795                 if (olen > 0)
796                         show_hex(data, olen, "  Auth Info");
797                 break;
798         }
799         case DHCPV6_OPT_UNICAST:
800                 if (olen >= sizeof (in6_addr_t))
801                         show_address("  Server Address", data);
802                 break;
803         case DHCPV6_OPT_STATUS_CODE:
804                 if (olen < sizeof (val16))
805                         break;
806                 (void) memcpy(&val16, data, sizeof (val16));
807                 val16 = ntohs(val16);
808                 (void) snprintf(get_line(0, 0), get_line_remain(),
809                     "  Status Code = %u (%s)", val16,
810                     status_to_str(val16));
811                 data += sizeof (val16);
812                 olen -= sizeof (val16);
813                 if (olen > 0)
814                         (void) snprintf(get_line(0, 0),
815                             get_line_remain(), "  Text = \"%.*s\"",
816                             olen, data);
817                 break;
818         case DHCPV6_OPT_VENDOR_CLASS:
819                 if (olen < sizeof (val32))
820                         break;
821                 (void) memcpy(&val32, data, sizeof (val32));
822                 data += sizeof (val32);
823                 olen -= sizeof (val32);
824                 val32 = ntohl(val32);
825                 (void) snprintf(get_line(0, 0), get_line_remain(),
826                     "  Enterprise Number = %lu (%s)", val32,
827                     entr_to_str(val32));
828                 /* FALLTHROUGH */
829         case DHCPV6_OPT_USER_CLASS:
830                 while (olen >= sizeof (val16)) {
831                         (void) memcpy(&val16, data, sizeof (val16));
832                         data += sizeof (val16);
833                         olen -= sizeof (val16);
834                         val16 = ntohs(val16);
835                         if (val16 > olen) {
836                                 (void) strlcpy(get_line(0, 0),
837                                     "  Truncated class",
838                                     get_line_remain());
839                                 val16 = olen;
840                         }
841                         show_hex(data, olen, "  Class");
842                         data += val16;
843                         olen -= val16;
844                 }
845                 break;
846         case DHCPV6_OPT_VENDOR_OPT: {
847                 dhcpv6_option_t sd6o;

849                 if (olen < sizeof (val32))
850                         break;
851                 (void) memcpy(&val32, data, sizeof (val32));
```



```
852                         data += sizeof (val32);
853                         olen -= sizeof (val32);
854                         val32 = ntohl(val32);
855                         (void) snprintf(get_line(0, 0), get_line_remain(),
856                             "  Enterprise Number = %lu (%s)", val32,
857                             entr_to_str(val32));
858                         while (olen >= sizeof (sd6o)) {
859                                 (void) memcpy(&sd6o, data, sizeof (sd6o));
860                                 sd6o.d6o_code = ntohs(sd6o.d6o_code);
861                                 sd6o.d6o_len = ntohs(sd6o.d6o_len);
862                                 (void) snprintf(get_line(0, 0),
863                                     get_line_remain(),
864                                     "  Vendor Option Code = %u", d6o.d6o_code);
865                                 data += sizeof (d6o);
866                                 olen -= sizeof (d6o);
867                                 if (sd6o.d6o_len > olen) {
868                                         (void) strlcpy(get_line(0, 0),
869                                             "  Vendor Option truncated",
870                                             get_line_remain());
871                                         sd6o.d6o_len = olen;
872                                 }
873                                 if (sd6o.d6o_len > 0) {
874                                         show_hex(data, sd6o.d6o_len,
875                                             "    Data");
876                                         data += sd6o.d6o_len;
877                                         olen -= sd6o.d6o_len;
878                                 }
879                         }
880                         break;
881                 }
882                 case DHCPV6_OPT_REMOTE_ID:
883                         if (olen < sizeof (val32))
884                                 break;
885                         (void) memcpy(&val32, data, sizeof (val32));
886                         data += sizeof (val32);
887                         olen -= sizeof (val32);
888                         val32 = ntohl(val32);
889                         (void) snprintf(get_line(0, 0), get_line_remain(),
890                             "  Enterprise Number = %lu (%s)", val32,
891                             entr_to_str(val32));
892                         /* FALLTHROUGH */
893                 case DHCPV6_OPT_INTERFACE_ID:
894                 case DHCPV6_OPT_SUBSCRIBER:
895                         if (olen > 0)
896                                 show_hex(data, olen, "  ID");
897                         break;
898                 case DHCPV6_OPT_RECONF_MSG:
899                         if (olen > 0) {
900                                 (void) snprintf(get_line(0, 0),
901                                     get_line_remain(),
902                                     "  Message Type = %u (%s)", *data,
903                                     reconf_to_str(*data));
904                         }
905                         break;
906                 case DHCPV6_OPT_SIP_NAMES:
907                 case DHCPV6_OPT_DNS_SEARCH:
908                 case DHCPV6_OPT_NIS_DOMAIN:
909                 case DHCPV6_OPT_BCMCS_SRV_D: {
910                         dhcp_symbol_t *symp;
911                         char *sp2;

913                         symp = inittab_getbycode(
914                             ITAB_CAT_STANDARD | ITAB_CAT_V6, ITAB_CONS_SNOOP,
915                             d6o.d6o_code);
916                         if (symp != NULL) {
917                                 str = inittab_decode(symp, data, olen, B_TRUE);
```

```
                                if (str != NULL) {
                                        sp = str;
                                        do {
                                                sp2 = strchr(sp, ' ');
                                                if (sp2 != NULL)
                                                        *sp2++ = '\0';
                                                (void) snprintf(get_line(0, 0),
                                                    get_line_remain(),
                                                    "  Name = %s", sp);
                                        } while ((sp = sp2) != NULL);
                                        free(str);
                                }
                                free(symp);
                        }
                        break;
                }
                case DHCPV6_OPT_SIP_ADDR:
                case DHCPV6_OPT_DNS_ADDR:
                case DHCPV6_OPT_NIS_SERVERS:
                case DHCPV6_OPT_SNTP_SERVERS:
                case DHCPV6_OPT_BCMCS_SRV_A:
                        while (olen >= sizeof (in6_addr_t)) {
                                show_address("  Address", data);
                                data += sizeof (in6_addr_t);
                                olen -= sizeof (in6_addr_t);
                        }
                        break;
                case DHCPV6_OPT_IAPREFIX: {
                        dhcpv6_iaprefix_t d6ip;

                        if (olen < DHCPV6_IAPREFIX_SIZE - sizeof (d6o))
                                break;
                        (void) memcpy(&d6ip, data - sizeof (d6o),
                            DHCPV6_IAPREFIX_SIZE);
                        data += DHCPV6_IAPREFIX_SIZE - sizeof (d6o);
                        olen -= DHCPV6_IAPREFIX_SIZE - sizeof (d6o);
                        show_address("  Prefix", d6ip.d6ip_addr);
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  Preferred lifetime = %u seconds",
                            ntohl(d6ip.d6ip_preflife));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  Valid lifetime = %u seconds",
                            ntohl(d6ip.d6ip_vallife));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  Prefix length = %u", d6ip.d6ip_preflen);
                        nest_options(data, olen, "ADDR: ", "Address");
                        break;
                }
                case DHCPV6_OPT_INFO_REFTIME:
                        if (olen < sizeof (val32))
                                break;
                        (void) memcpy(&val32, data, sizeof (val32));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  Refresh Time = %lu seconds", ntohl(val32));
                        break;
                case DHCPV6_OPT_GEOCONF_CVC: {
                        dhcpv6_civic_t d6c;
                        int solen;

                        if (olen < DHCPV6_CIVIC_SIZE - sizeof (d6o))
                                break;
                        (void) memcpy(&d6c, data - sizeof (d6o),
                            DHCPV6_CIVIC_SIZE);
                        data += DHCPV6_CIVIC_SIZE - sizeof (d6o);
                        olen -= DHCPV6_CIVIC_SIZE - sizeof (d6o);
                        (void) snprintf(get_line(0, 0), get_line_remain(),
```



```
                            "  What Location = %u (%s)", d6c.d6c_what,
                            cwhat_to_str(d6c.d6c_what));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  Country Code = %.*s", sizeof (d6c.d6c_cc),
                            d6c.d6c_cc);
                        while (olen >= 2) {
                                (void) snprintf(get_line(0, 0),
                                    get_line_remain(),
                                    "  CA Element = %u (%s)", *data,
                                    catype_to_str(*data));
                                solen = data[1];
                                data += 2;
                                olen -= 2;
                                if (solen > olen) {
                                        (void) strlcpy(get_line(0, 0),
                                            "  CA Element truncated",
                                            get_line_remain());
                                        solen = olen;
                                }
                                if (solen > 0) {
                                        show_ascii(data, solen, "  CA Data");
                                        data += solen;
                                        olen -= solen;
                                }
                        }
                        break;
                }
                case DHCPV6_OPT_CLIENT_FQDN: {
                        dhcp_symbol_t *symp;

                        if (olen == 0)
                                break;
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  Flags = %02x", *data);
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "        %s", getflag(*data, DHCPV6_FQDNF_S,
                            "Perform AAAA RR updates", "No AAAA RR updates"));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "        %s", getflag(*data, DHCPV6_FQDNF_O,
                            "Server override updates",
                            "No server override updates"));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "        %s", getflag(*data, DHCPV6_FQDNF_N,
                            "Server performs no updates",
                            "Server performs updates"));
                        symp = inittab_getbycode(
                            ITAB_CAT_STANDARD | ITAB_CAT_V6, ITAB_CONS_SNOOP,
                            d6o.d6o_code);
                        if (symp != NULL) {
                                str = inittab_decode(symp, data, olen, B_TRUE);
                                if (str != NULL) {
                                        (void) snprintf(get_line(0, 0),
                                            get_line_remain(),
                                            "  FQDN = %s", str);
                                        free(str);
                                }
                                free(symp);
                        }
                        break;
                }
                }
                data = ostart + d6o.d6o_len;
                len -= d6o.d6o_len;
        }
        if (len != 0) {
                (void) strlcpy(get_line(0, 0), "Option entry truncated",
```

```
1050                     get_line_remain());
1051         }
1052 }
```

_____***unchanged_portion_omitted_***