```
**********************************************************
    6577 Tue Feb 11 10:48:28 2014
new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_arp.c
4587 snoop misdecodes DHCPv6 DHCPV6_DUID_LL identifiers
**********************************************************
/*
 * 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 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/errno.h>
#include <setjmp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#endif /* ! codereview */
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netdb.h>
#include <net/if_types.h>
#include <arpa/inet.h>
#endif /* ! codereview */

#include "snoop.h"

extern char *dlc_header;
extern jmp_buf xdr_err;

static char *printip(unsigned char *);
static char *addrtoname_align(unsigned char *);

static char unarp_addr[] = "Unknown";
char *opname[] = {
        "",
        "ARP Request",
        "ARP Reply",
        "REVARP Request",
        "REVARP Reply",
```

```
};

void
interpret_arp(int flags, struct arphdr *ap, int alen)
{
        char *line;
        extern char *src_name, *dst_name;
        unsigned char *sip, *tip, *sha, *tha;
        char *smacbuf = NULL, *dmacbuf = NULL;
        int maclen;
        ushort_t arpop;
        boolean_t is_ip = B_FALSE;

        /*
         * Check that at least the generic ARP header was received.
         */
        if (sizeof (struct arphdr) > alen)
                goto short_packet;

        arpop = ntohs(ap->ar_op);
        maclen = ap->ar_hln;
        if (ntohs(ap->ar_pro) == ETHERTYPE_IP)
                is_ip = B_TRUE;

        sha = (unsigned char *)(ap + 1);
        sip = sha + maclen;
        tha = sip + ap->ar_pln;
        tip = tha + maclen;

        /*
         * Check that the protocol/hardware addresses were received.
         */
        if ((tip + ap->ar_pln) > ((unsigned char *)ap + alen))
                goto short_packet;

        if (maclen == 0) {
                smacbuf = dmacbuf = unarp_addr;
        } else {
                if (((flags & F_DTAIL) && is_ip) || (arpop == ARPOP_REPLY)) {
                        smacbuf = _link_ntoa(sha, NULL, maclen, IFT_OTHER);
                        if (smacbuf == NULL)
                                pr_err("Warning: malloc failure");
                }

                if (((flags & F_DTAIL) && is_ip) || (arpop ==
                    REVARP_REQUEST) || (arpop == REVARP_REPLY)) {
                        dmacbuf = _link_ntoa(tha, NULL, maclen, IFT_OTHER);
                        if (dmacbuf == NULL)
                                pr_err("Warning: malloc failure");
                }
        }

        src_name = addrtoname_align(sip);

        if (flags & F_SUM) {

                line = get_sum_line();

                switch (arpop) {
                case ARPOP_REQUEST:
                        (void) snprintf(line, MAXLINE, "ARP C Who is %s ?",
                            printip(tip));
                        break;
                case ARPOP_REPLY:
                        (void) snprintf(line, MAXLINE, "ARP R %s is %s",
                            printip(sip), smacbuf);
                        dst_name = addrtoname_align(tip);
                        break;
                case REVARP_REQUEST:
                        (void) snprintf(line, MAXLINE, "RARP C Who is %s ?",
                            dmacbuf);
                        break;
                case REVARP_REPLY:
                        (void) snprintf(line, MAXLINE, "RARP R %s is %s",
                            dmacbuf, printip(tip));
                        dst_name = addrtoname_align(tip);
                        break;
                }
        }

        if (flags & F_DTAIL) {
                show_header("ARP:  ", "ARP/RARP Frame", alen);
                show_space();
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Hardware type = %d (%s)", ntohs(ap->ar_hrd),
                    arp_htype(ntohs(ap->ar_hrd)));
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Protocol type = %04x (%s)", ntohs(ap->ar_pro),
                    print_ethertype(ntohs(ap->ar_pro)));
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Length of hardware address = %d bytes", ap->ar_hln);
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Length of protocol address = %d bytes", ap->ar_pln);
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Opcode %d (%s)", arpop,
                    (arpop > REVARP_REPLY) ? opname[0] : opname[arpop]);

                if (is_ip) {
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "Sender’s hardware address = %s", smacbuf);
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "Sender’s protocol address = %s",
                            printip(sip));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "Target hardware address = %s",
                            arpop == ARPOP_REQUEST ? "?" : dmacbuf);
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "Target protocol address = %s",
                            arpop == REVARP_REQUEST ? "?" :
                            printip(tip));
                }
                show_trailer();
        }

        if (maclen != 0) {
                free(smacbuf);
                free(dmacbuf);
        }
        return;

short_packet:
        if (flags & F_SUM) {
                (void) snprintf(get_sum_line(), MAXLINE,
                    "ARP (short packet)");
        } else if (flags & F_DTAIL) {
                show_header("ARP:  ", "ARP/RARP Frame", alen);
                show_space();
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "ARP (short packet)");
        }
}
```

```
char *
printip(unsigned char *p)
{
        static char buff[MAXHOSTNAMELEN + 32];
        char *ap, *np;
        struct in_addr a;

        memcpy(&a, p, 4);
        ap = (char *)inet_ntoa(a);
        np = (char *)addrtoname(AF_INET, &a);
        (void) snprintf(buff, MAXHOSTNAMELEN, "%s, %s", ap, np);
        return (buff);
}

char *
addrtoname_align(unsigned char *p)
{
        struct in_addr a;

        memcpy(&a, p, 4);
        return ((char *)addrtoname(AF_INET, &a));
}

/*
 * These numbers are assigned by the IANA.  See the arp-parameters registry.
 * Only those values that are used within Solaris have #defines.
 */
const char *
arp_htype(int t)
{
        switch (t) {
        case ARPHRD_ETHER:
                return ("Ethernet (10Mb)");
        case 2:
                return ("Experimental Ethernet (3Mb)");
                return ("Experimental Ethernet (3MB)");
        case 3:
                return ("Amateur Radio AX.25");
        case 4:
                return ("Proteon ProNET Token Ring");
        case 5:
                return ("Chaos");
        case ARPHRD_IEEE802:
                return ("IEEE 802");
        case 7:
                return ("ARCNET");
        case 8:
                return ("Hyperchannel");
        case 9:
                return ("Lanstar");
        case 10:
                return ("Autonet");
        case 11:
                return ("LocalTalk");
        case 12:
                return ("LocalNet");
        case 13:
                return ("Ultra Link");
        case 14:
                return ("SMDS");
        case ARPHRD_FRAME:
                return ("Frame Relay");
        case ARPHRD_ATM:
                return ("ATM");
        case ARPHRD_HDLC:
                return ("HDLC");
        case ARPHRD_FC:
                return ("Fibre Channel");
        case ARPHRD_IPATM:
                return ("IP-ATM");
        case ARPHRD_METRICOM:
                return ("Metricom");
#endif /* ! codereview */
        case ARPHRD_TUNNEL:
                return ("Tunnel");
        case ARPHRD_IB:
                return ("IPIB");
        };
        return ("UNKNOWN");
}
```

```
**********************************************************
   28711 Tue Feb 11 10:48:29 2014
new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_dhcpv6.c
4587 snoop misdecodes DHCPv6 DHCPV6_DUID_LL identifiers
**********************************************************
/*
 * 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);
                d6o = dhcpv6_find_option(data, len, NULL,
```

```c
                    DHCPV6_OPT_STATUS_CODE, &optlen);
                optlen -= sizeof (*d6o);
                if (d6o != NULL && optlen >= sizeof (statuscode)) {
                        (void) memcpy(&statuscode, d6o + 1,
                            sizeof (statuscode));
                        line += snprintf(line, MAXLINE - (line - lstart),
                            " status=%u", ntohs(statuscode));
                        optlen -= sizeof (statuscode);
                        if (optlen > 0) {
                                line += snprintf(line,
                                    MAXLINE - (line - lstart), " \"%.*s\"",
                                    optlen, (char *)(d6o + 1) + 2);
                        }
                }
                d6o = dhcpv6_find_option(data, len, NULL,
                    DHCPV6_OPT_RELAY_MSG, &optlen);
                optlen -= sizeof (*d6o);
                if (d6o != NULL && optlen >= 1) {
                        line += snprintf(line, MAXLINE - (line - lstart),
                            " relay=%s", mtype_to_str(*(uint8_t *)(d6o + 1)));
                }
        } else if (flags & F_DTAIL) {
                show_header("DHCPv6: ",
                    "Dynamic Host Configuration Protocol Version 6", len);
                show_space();
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Message type (msg-type) = %u (%s)", data[0],
                    mtype_to_str(data[0]));
                if (data[0] == DHCPV6_MSG_RELAY_FORW ||
                    data[0] == DHCPV6_MSG_RELAY_REPL) {
                        if (len < sizeof (d6r)) {
                                (void) strlcpy(get_line(0, 0), "Truncated",
                                    get_line_remain());
                                return (olen);
                        }
                        (void) memcpy(&d6r, data, sizeof (d6r));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "Hop count = %u", d6r.d6r_hop_count);
                        show_address("Link address", d6r.d6r_linkaddr);
                        show_address("Peer address", d6r.d6r_peeraddr);
                        data += sizeof (d6r);
                        len -= sizeof (d6r);
                } else {
                        if (len < sizeof (d6m)) {
                                (void) strlcpy(get_line(0, 0), "Truncated",
                                    get_line_remain());
                                return (olen);
                        }
                        (void) memcpy(&d6m, data, sizeof (d6m));
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "Transaction ID = %x", DHCPV6_GET_TRANSID(&d6m));
                        data += sizeof (d6m);
                        len -= sizeof (d6m);
                }
                show_space();
                show_options(data, len);
                show_space();
        }
        return (olen);
}

static const char *
mtype_to_str(uint8_t mtype)
{
        switch (mtype) {
        case DHCPV6_MSG_SOLICIT:
```

```c
                return ("Solicit");
        case DHCPV6_MSG_ADVERTISE:
                return ("Advertise");
        case DHCPV6_MSG_REQUEST:
                return ("Request");
        case DHCPV6_MSG_CONFIRM:
                return ("Confirm");
        case DHCPV6_MSG_RENEW:
                return ("Renew");
        case DHCPV6_MSG_REBIND:
                return ("Rebind");
        case DHCPV6_MSG_REPLY:
                return ("Reply");
        case DHCPV6_MSG_RELEASE:
                return ("Release");
        case DHCPV6_MSG_DECLINE:
                return ("Decline");
        case DHCPV6_MSG_RECONFIGURE:
                return ("Reconfigure");
        case DHCPV6_MSG_INFO_REQ:
                return ("Information-Request");
        case DHCPV6_MSG_RELAY_FORW:
                return ("Relay-Forward");
        case DHCPV6_MSG_RELAY_REPL:
                return ("Relay-Reply");
        default:
                return ("Unknown");
        }
}

static const char *
option_to_str(uint8_t mtype)
{
        switch (mtype) {
        case DHCPV6_OPT_CLIENTID:
                return ("Client Identifier");
        case DHCPV6_OPT_SERVERID:
                return ("Server Identifier");
        case DHCPV6_OPT_IA_NA:
                return ("Identity Association for Non-temporary Addresses");
        case DHCPV6_OPT_IA_TA:
                return ("Identity Association for Temporary Addresses");
        case DHCPV6_OPT_IAADDR:
                return ("IA Address");
        case DHCPV6_OPT_ORO:
                return ("Option Request");
        case DHCPV6_OPT_PREFERENCE:
                return ("Preference");
        case DHCPV6_OPT_ELAPSED_TIME:
                return ("Elapsed Time");
        case DHCPV6_OPT_RELAY_MSG:
                return ("Relay Message");
        case DHCPV6_OPT_AUTH:
                return ("Authentication");
        case DHCPV6_OPT_UNICAST:
                return ("Server Unicast");
        case DHCPV6_OPT_STATUS_CODE:
                return ("Status Code");
        case DHCPV6_OPT_RAPID_COMMIT:
                return ("Rapid Commit");
        case DHCPV6_OPT_USER_CLASS:
                return ("User Class");
        case DHCPV6_OPT_VENDOR_CLASS:
                return ("Vendor Class");
        case DHCPV6_OPT_VENDOR_OPT:
                return ("Vendor-specific Information");
```

```
	case DHCPV6_OPT_INTERFACE_ID:
		return ("Interface-Id");
	case DHCPV6_OPT_RECONF_MSG:
		return ("Reconfigure Message");
	case DHCPV6_OPT_RECONF_ACC:
		return ("Reconfigure Accept");
	case DHCPV6_OPT_SIP_NAMES:
		return ("SIP Servers Domain Name List");
	case DHCPV6_OPT_SIP_ADDR:
		return ("SIP Servers IPv6 Address List");
	case DHCPV6_OPT_DNS_ADDR:
		return ("DNS Recursive Name Server");
	case DHCPV6_OPT_DNS_SEARCH:
		return ("Domain Search List");
	case DHCPV6_OPT_IA_PD:
		return ("Identity Association for Prefix Delegation");
	case DHCPV6_OPT_IAPREFIX:
		return ("IA_PD Prefix");
	case DHCPV6_OPT_NIS_SERVERS:
		return ("Network Information Service Servers");
	case DHCPV6_OPT_NIS_DOMAIN:
		return ("Network Information Service Domain Name");
	case DHCPV6_OPT_SNTP_SERVERS:
		return ("Simple Network Time Protocol Servers");
	case DHCPV6_OPT_INFO_REFTIME:
		return ("Information Refresh Time");
	case DHCPV6_OPT_BCMCS_SRV_D:
		return ("BCMCS Controller Domain Name List");
	case DHCPV6_OPT_BCMCS_SRV_A:
		return ("BCMCS Controller IPv6 Address");
	case DHCPV6_OPT_GEOCONF_CVC:
		return ("Civic Location");
	case DHCPV6_OPT_REMOTE_ID:
		return ("Relay Agent Remote-ID");
	case DHCPV6_OPT_SUBSCRIBER:
		return ("Relay Agent Subscriber-ID");
	case DHCPV6_OPT_CLIENT_FQDN:
		return ("Client FQDN");
	default:
		return ("Unknown");
	}
}

static const char *
duidtype_to_str(uint16_t dtype)
{
	switch (dtype) {
	case DHCPV6_DUID_LLT:
		return ("Link-layer Address Plus Time");
	case DHCPV6_DUID_EN:
		return ("Enterprise Number");
	case DHCPV6_DUID_LL:
		return ("Link-layer Address");
	default:
		return ("Unknown");
	}
}

static const char *
status_to_str(uint16_t status)
{
	switch (status) {
	case DHCPV6_STAT_SUCCESS:
		return ("Success");
	case DHCPV6_STAT_UNSPECFAIL:
		return ("Failure, reason unspecified");
```

```
	case DHCPV6_STAT_NOADDRS:
		return ("No addresses for IAs");
	case DHCPV6_STAT_NOBINDING:
		return ("Client binding unavailable");
	case DHCPV6_STAT_NOTONLINK:
		return ("Prefix not on link");
	case DHCPV6_STAT_USEMCAST:
		return ("Use multicast");
	case DHCPV6_STAT_NOPREFIX:
		return ("No prefix available");
	default:
		return ("Unknown");
	}
}

static const char *
entr_to_str(uint32_t entr)
{
	switch (entr) {
	case DHCPV6_SUN_ENT:
		return ("Sun Microsystems");
	default:
		return ("Unknown");
	}
}

static const char *
reconf_to_str(uint8_t msgtype)
{
	switch (msgtype) {
	case DHCPV6_RECONF_RENEW:
		return ("Renew");
	case DHCPV6_RECONF_INFO:
		return ("Information-request");
	default:
		return ("Unknown");
	}
}

static const char *
authproto_to_str(uint8_t aproto)
{
	switch (aproto) {
	case DHCPV6_PROTO_DELAYED:
		return ("Delayed");
	case DHCPV6_PROTO_RECONFIG:
		return ("Reconfigure Key");
	default:
		return ("Unknown");
	}
}

static const char *
authalg_to_str(uint8_t aproto, uint8_t aalg)
{
	switch (aproto) {
	case DHCPV6_PROTO_DELAYED:
	case DHCPV6_PROTO_RECONFIG:
		switch (aalg) {
		case DHCPV6_ALG_HMAC_MD5:
			return ("HMAC-MD5 Signature");
		default:
			return ("Unknown");
		}
		break;
	default:
		return ("Unknown");
	}
}

static const char *
authrdm_to_str(uint8_t ardm)
{
	switch (ardm) {
	case DHCPV6_RDM_MONOCNT:
		return ("Monotonic Counter");
	default:
		return ("Unknown");
	}
}

static const char *
cwhat_to_str(uint8_t what)
{
	switch (what) {
	case DHCPV6_CWHAT_SERVER:
		return ("Server");
	case DHCPV6_CWHAT_NETWORK:
		return ("Network");
	case DHCPV6_CWHAT_CLIENT:
		return ("Client");
	default:
		return ("Unknown");
	}
}

static const char *
catype_to_str(uint8_t catype)
{
	switch (catype) {
	case CIVICADDR_LANG:
		return ("Language; RFC 2277");
	case CIVICADDR_A1:
		return ("National division (state)");
	case CIVICADDR_A2:
		return ("County");
	case CIVICADDR_A3:
		return ("City");
	case CIVICADDR_A4:
		return ("City division");
	case CIVICADDR_A5:
		return ("Neighborhood");
	case CIVICADDR_A6:
		return ("Street group");
	case CIVICADDR_PRD:
		return ("Leading street direction");
	case CIVICADDR_POD:
		return ("Trailing street suffix");
	case CIVICADDR_STS:
		return ("Street suffix or type");
	case CIVICADDR_HNO:
		return ("House number");
	case CIVICADDR_HNS:
		return ("House number suffix");
	case CIVICADDR_LMK:
		return ("Landmark");
	case CIVICADDR_LOC:
		return ("Additional location information");
	case CIVICADDR_NAM:
		return ("Name/occupant");
	case CIVICADDR_PC:
		return ("Postal Code/ZIP");
```

```
	case CIVICADDR_BLD:
		return ("Building");
	case CIVICADDR_UNIT:
		return ("Unit/apt/suite");
	case CIVICADDR_FLR:
		return ("Floor");
	case CIVICADDR_ROOM:
		return ("Room number");
	case CIVICADDR_TYPE:
		return ("Place type");
	case CIVICADDR_PCN:
		return ("Postal community name");
	case CIVICADDR_POBOX:
		return ("Post office box");
	case CIVICADDR_ADDL:
		return ("Additional code");
	case CIVICADDR_SEAT:
		return ("Seat/desk");
	case CIVICADDR_ROAD:
		return ("Primary road or street");
	case CIVICADDR_RSEC:
		return ("Road section");
	case CIVICADDR_RBRA:
		return ("Road branch");
	case CIVICADDR_RSBR:
		return ("Road sub-branch");
	case CIVICADDR_SPRE:
		return ("Street name pre-modifier");
	case CIVICADDR_SPOST:
		return ("Street name post-modifier");
	case CIVICADDR_SCRIPT:
		return ("Script");
	default:
		return ("Unknown");
	}
}

static void
show_hex(const uint8_t *data, int len, const char *name)
{
	char buffer[16 * 3 + 1];
	int nlen;
	int i;
	char sep;

	nlen = strlen(name);
	sep = '=';
	while (len > 0) {
		for (i = 0; i < 16 && i < len; i++)
			(void) snprintf(buffer + 3 * i, 4, " %02x", *data++);
		(void) snprintf(get_line(0, 0), get_line_remain(), "%*s %c%s",
		    nlen, name, sep, buffer);
		name = "";
		sep = ' ';
		len -= i;
	}
}

static void
show_ascii(const uint8_t *data, int len, const char *name)
{
	char buffer[64], *bp;
	int nlen;
	int i;
	char sep;
```

```c
        nlen = strlen(name);
        sep = '=';
        while (len > 0) {
                bp = buffer;
                for (i = 0; i < sizeof (buffer) - 4 && len > 0; len--) {
                        if (!isascii(*data) || !isprint(*data))
                                bp += snprintf(bp, 5, "\\%03o", *data++);
                        else
                                *bp++;
                }
                *bp = '\0';
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "%*s %c \"%s\"", nlen, name, sep, buffer);
                sep = ' ';
                name = "";
        }
}

static void
show_address(const char *addrname, const void *aptr)
{
        char *hname;
        char addrstr[INET6_ADDRSTRLEN];
        in6_addr_t addr;

        (void) memcpy(&addr, aptr, sizeof (in6_addr_t));
        (void) inet_ntop(AF_INET6, &addr, addrstr, sizeof (addrstr));
        hname = addrtoname(AF_INET6, &addr);
        if (strcmp(hname, addrstr) == 0) {
                (void) snprintf(get_line(0, 0), get_line_remain(), "%s = %s",
                    addrname, addrstr);
        } else {
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "%s = %s (%s)", addrname, addrstr, hname);
        }
}

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

        if (olen <= 0)
                return;
        oldprefix = prot_prefix;
        oldnest = prot_nest_prefix;
        str = malloc(strlen(prot_nest_prefix) + strlen(prot_prefix) + 1);
        if (str == NULL) {
                prot_nest_prefix = prot_prefix;
        } else {
                (void) sprintf(str, "%s%s", prot_nest_prefix, prot_prefix);
                prot_nest_prefix = str;
        }
        show_header(prefix, title, 0);
        show_options(data, olen);
        free(str);
        prot_prefix = oldprefix;
        prot_nest_prefix = oldnest;
}

static void
show_options(const uint8_t *data, int len)
{
        dhcpv6_option_t d6o;
        uint_t olen, retlen;
        uint16_t val16;
```

```c
        uint16_t type;
        uint32_t val32;
        const uint8_t *ostart;
        char *str, *sp;
        char *oldnest;

        /*
         * Be very careful with negative numbers; ANSI signed/unsigned
         * comparison doesn't work as expected.
         */
        while (len >= (signed)sizeof (d6o)) {
                (void) memcpy(&d6o, data, sizeof (d6o));
                d6o.d6o_code = ntohs(d6o.d6o_code);
                d6o.d6o_len = olen = ntohs(d6o.d6o_len);
                (void) snprintf(get_line(0, 0), get_line_remain(),
                    "Option Code = %u (%s)", d6o.d6o_code,
                    option_to_str(d6o.d6o_code));
                ostart = data += sizeof (d6o);
                len -= sizeof (d6o);
                if (olen > len) {
                        (void) strlcpy(get_line(0, 0), "Option truncated",
                            get_line_remain());
                        olen = len;
                }
                switch (d6o.d6o_code) {
                case DHCPV6_OPT_CLIENTID:
                case DHCPV6_OPT_SERVERID:
                        if (olen < sizeof (val16))
                                break;
                        (void) memcpy(&val16, data, sizeof (val16));
                        data += sizeof (val16);
                        olen -= sizeof (val16);
                        type = ntohs(val16);
                        (void) snprintf(get_line(0, 0), get_line_remain(),
                            "  DUID Type = %u (%s)", type,
                            duidtype_to_str(type));
                        if (type == DHCPV6_DUID_LLT || type == DHCPV6_DUID_LL) {
                                if (olen < sizeof (val16))
                                        break;
                                (void) memcpy(&val16, data, sizeof (val16));
                                data += sizeof (val16);
                                olen -= sizeof (val16);
                                val16 = ntohs(val16);
                                (void) snprintf(get_line(0, 0),
                                    get_line_remain(),
                                    "  Hardware Type = %u (%s)", val16,
                                    arp_htype(val16));
                                    arp_htype(type));
                        }
                        if (type == DHCPV6_DUID_LLT) {
                                time_t timevalue;

                                if (olen < sizeof (val32))
                                        break;
                                (void) memcpy(&val32, data, sizeof (val32));
                                data += sizeof (val32);
                                olen -= sizeof (val32);
                                timevalue = ntohl(val32) + DUID_TIME_BASE;
                                (void) snprintf(get_line(0, 0),
                                    get_line_remain(),
                                    "  Time = %lu (%.24s)", ntohl(val32),
                                    ctime(&timevalue));
                        }
                        if (type == DHCPV6_DUID_EN) {
                                if (olen < sizeof (val32))
                                        break;
```

```
				(void) memcpy(&val32, data, sizeof (val32));
				data += sizeof (val32);
				olen -= sizeof (val32);
				val32 = ntohl(val32);
				(void) snprintf(get_line(0, 0),
				    get_line_remain(),
				    "  Enterprise Number = %lu (%s)", val32,
				    entr_to_str(val32));
			}
			if (olen == 0)
				break;
			if ((str = malloc(olen * 3)) == NULL)
				pr_err("interpret_dhcpv6: no mem");
			sp = str + snprintf(str, 3, "%02x", *data++);
			while (--olen > 0) {
				*sp++ = (type == DHCPV6_DUID_LLT ||
				    type == DHCPV6_DUID_LL) ? ':' : ' ';
				sp = sp + snprintf(sp, 3, "%02x", *data++);
			}
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    (type == DHCPV6_DUID_LLT ||
			    type == DHCPV6_DUID_LL) ?
			    "  Link Layer Address = %s" :
			    "  Identifier = %s", str);
			free(str);
			break;
		case DHCPV6_OPT_IA_NA:
		case DHCPV6_OPT_IA_PD: {
			dhcpv6_ia_na_t d6in;

			if (olen < sizeof (d6in) - sizeof (d6o))
				break;
			(void) memcpy(&d6in, data - sizeof (d6o),
			    sizeof (d6in));
			data += sizeof (d6in) - sizeof (d6o);
			olen -= sizeof (d6in) - sizeof (d6o);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  IAID = %u", ntohl(d6in.d6in_iaid));
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  T1 (renew) = %u seconds", ntohl(d6in.d6in_t1));
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  T2 (rebind) = %u seconds", ntohl(d6in.d6in_t2));
			nest_options(data, olen, "IA: ",
			    "Identity Association");
			break;
		}
		case DHCPV6_OPT_IA_TA: {
			dhcpv6_ia_ta_t d6it;

			if (olen < sizeof (d6it) - sizeof (d6o))
				break;
			(void) memcpy(&d6it, data - sizeof (d6o),
			    sizeof (d6it));
			data += sizeof (d6it) - sizeof (d6o);
			olen -= sizeof (d6it) - sizeof (d6o);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  IAID = %u", ntohl(d6it.d6it_iaid));
			nest_options(data, olen, "IA: ",
			    "Identity Association");
			break;
		}
		case DHCPV6_OPT_IAADDR: {
			dhcpv6_iaaddr_t d6ia;

			if (olen < sizeof (d6ia) - sizeof (d6o))
				break;
```

```
			(void) memcpy(&d6ia, data - sizeof (d6o),
			    sizeof (d6ia));
			data += sizeof (d6ia) - sizeof (d6o);
			olen -= sizeof (d6ia) - sizeof (d6o);
			show_address("  Address", &d6ia.d6ia_addr);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Preferred lifetime = %u seconds",
			    ntohl(d6ia.d6ia_preflife));
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Valid lifetime = %u seconds",
			    ntohl(d6ia.d6ia_vallife));
			nest_options(data, olen, "ADDR: ", "Address");
			break;
		}
		case DHCPV6_OPT_ORO:
			while (olen >= sizeof (val16)) {
				(void) memcpy(&val16, data, sizeof (val16));
				val16 = ntohs(val16);
				(void) snprintf(get_line(0, 0),
				    get_line_remain(),
				    "  Requested Option Code = %u (%s)", val16,
				    option_to_str(val16));
				data += sizeof (val16);
				olen -= sizeof (val16);
			}
			break;
		case DHCPV6_OPT_PREFERENCE:
			if (olen > 0) {
				(void) snprintf(get_line(0, 0),
				    get_line_remain(),
				    *data == 255 ?
				    "  Preference = %u (immediate)" :
				    "  Preference = %u", *data);
			}
			break;
		case DHCPV6_OPT_ELAPSED_TIME:
			if (olen == sizeof (val16)) {
				(void) memcpy(&val16, data, sizeof (val16));
				val16 = ntohs(val16);
				(void) snprintf(get_line(0, 0),
				    get_line_remain(),
				    "  Elapsed Time = %u.%02u seconds",
				    val16 / 100, val16 % 100);
			}
			break;
		case DHCPV6_OPT_RELAY_MSG:
			if (olen > 0) {
				oldnest = prot_nest_prefix;
				prot_nest_prefix = prot_prefix;
				retlen = interpret_dhcpv6(F_DTAIL, data, olen);
				prot_prefix = prot_nest_prefix;
				prot_nest_prefix = oldnest;
			}
			break;
		case DHCPV6_OPT_AUTH: {
			dhcpv6_auth_t d6a;

			if (olen < DHCPV6_AUTH_SIZE - sizeof (d6o))
				break;
			(void) memcpy(&d6a, data - sizeof (d6o),
			    DHCPV6_AUTH_SIZE);
			data += DHCPV6_AUTH_SIZE - sizeof (d6o);
			olen += DHCPV6_AUTH_SIZE - sizeof (d6o);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Protocol = %u (%s)", d6a.d6a_proto,
			    authproto_to_str(d6a.d6a_proto));
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Algorithm = %u (%s)", d6a.d6a_alg,
			    authalg_to_str(d6a.d6a_proto, d6a.d6a_alg));
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Replay Detection Method = %u (%s)", d6a.d6a_rdm,
			    authrdm_to_str(d6a.d6a_rdm));
			show_hex(d6a.d6a_replay, sizeof (d6a.d6a_replay),
			    "  RDM Data");
			if (olen > 0)
				show_hex(data, olen, "  Auth Info");
			break;
		}
		case DHCPV6_OPT_UNICAST:
			if (olen >= sizeof (in6_addr_t))
				show_address("  Server Address", data);
			break;
		case DHCPV6_OPT_STATUS_CODE:
			if (olen < sizeof (val16))
				break;
			(void) memcpy(&val16, data, sizeof (val16));
			val16 = ntohs(val16);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Status Code = %u (%s)", val16,
			    status_to_str(val16));
			data += sizeof (val16);
			olen -= sizeof (val16);
			if (olen > 0)
				(void) snprintf(get_line(0, 0),
				    get_line_remain(), "  Text = \"%.*s\"",
				    olen, data);
			break;
		case DHCPV6_OPT_VENDOR_CLASS:
			if (olen < sizeof (val32))
				break;
			(void) memcpy(&val32, data, sizeof (val32));
			data += sizeof (val32);
			olen -= sizeof (val32);
			val32 = ntohl(val32);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Enterprise Number = %lu (%s)", val32,
			    entr_to_str(val32));
			/* FALLTHROUGH */
		case DHCPV6_OPT_USER_CLASS:
			while (olen >= sizeof (val16)) {
				(void) memcpy(&val16, data, sizeof (val16));
				data += sizeof (val16);
				olen -= sizeof (val16);
				val16 = ntohs(val16);
				if (val16 > olen) {
					(void) strlcpy(get_line(0, 0),
					    "  Truncated class",
					    get_line_remain());
					val16 = olen;
				}
				show_hex(data, olen, "  Class");
				data += val16;
				olen -= val16;
			}
			break;
		case DHCPV6_OPT_VENDOR_OPT: {
			dhcpv6_option_t sd6o;

			if (olen < sizeof (val32))
				break;
			(void) memcpy(&val32, data, sizeof (val32));
			data += sizeof (val32);
```

```
			olen -= sizeof (val32);
			val32 = ntohl(val32);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Enterprise Number = %lu (%s)", val32,
			    entr_to_str(val32));
			while (olen >= sizeof (sd6o)) {
				(void) memcpy(&sd6o, data, sizeof (sd6o));
				sd6o.d6o_code = ntohs(sd6o.d6o_code);
				sd6o.d6o_len = ntohs(sd6o.d6o_len);
				(void) snprintf(get_line(0, 0),
				    get_line_remain(),
				    "  Vendor Option Code = %u", d6o.d6o_code);
				data += sizeof (d6o);
				olen -= sizeof (d6o);
				if (sd6o.d6o_len > olen) {
					(void) strlcpy(get_line(0, 0),
					    "  Vendor Option truncated",
					    get_line_remain());
					sd6o.d6o_len = olen;
				}
				if (sd6o.d6o_len > 0) {
					show_hex(data, sd6o.d6o_len,
					    "    Data");
					data += sd6o.d6o_len;
					olen -= sd6o.d6o_len;
				}
			}
			break;
		}
		case DHCPV6_OPT_REMOTE_ID:
			if (olen < sizeof (val32))
				break;
			(void) memcpy(&val32, data, sizeof (val32));
			data += sizeof (val32);
			olen -= sizeof (val32);
			val32 = ntohl(val32);
			(void) snprintf(get_line(0, 0), get_line_remain(),
			    "  Enterprise Number = %lu (%s)", val32,
			    entr_to_str(val32));
			/* FALLTHROUGH */
		case DHCPV6_OPT_INTERFACE_ID:
		case DHCPV6_OPT_SUBSCRIBER:
			if (olen > 0)
				show_hex(data, olen, "  ID");
			break;
		case DHCPV6_OPT_RECONF_MSG:
			if (olen > 0) {
				(void) snprintf(get_line(0, 0),
				    get_line_remain(),
				    "  Message Type = %u (%s)", *data,
				    reconf_to_str(*data));
			}
			break;
		case DHCPV6_OPT_SIP_NAMES:
		case DHCPV6_OPT_DNS_SEARCH:
		case DHCPV6_OPT_NIS_DOMAIN:
		case DHCPV6_OPT_BCMCS_SRV_D: {
			dhcp_symbol_t *symp;
			char *sp2;

			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) {
```

```
                                        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",
                    get_line_remain());
        }
}
```
______*unchanged_portion_omitted_*