sunghan-chang wpa_supplicant: move to external ... History

1 contributor

359 lines (312 sloc) | 9.41 KB ...

```
1  /*
2   * WPA Supplicant - Layer2 packet handling with libpcap/libdnet and WinPcap
3   * Copyright (c) 2003-2006, Jouni Malinen <j@w1.fi>
4   *
5   * This software may be distributed under the terms of the BSD license.
6   * See README for more details.
7   */
8
9  #include "includes.h"
10 #ifndef CONFIG_NATIVE_WINDOWS
11 #include <sys/ioctl.h>
12 #endif                                          /* CONFIG_NATIVE_WINDOWS */
13 #include <pcap.h>
14 #ifndef CONFIG_WINPCAP
15 #include <dnet.h>
16 #endif                                          /* CONFIG_WINPCAP */
17
18 #include "common.h"
19 #include "eloop.h"
20 #include "l2_packet.h"
21
22 static const u8 pae_group_addr[ETH_ALEN] = { 0x01, 0x80, 0xc2, 0x00, 0x00, 0x03 };
23
24 struct l2_packet_data {
25         pcap_t *pcap;
26 #ifdef CONFIG_WINPCAP
27         unsigned int num_fast_poll;
28 #else                                           /* CONFIG_WINPCAP */
29         eth_t *eth;
```

```
#endif                                                  /* CONFIG_WINPCAP */
        char ifname[100];
        u8 own_addr[ETH_ALEN];
        void (*rx_callback)(void *ctx, const u8 *src_addr, const u8 *buf, size_t len);
        void *rx_callback_ctx;
        int l2_hdr;                                     /* whether to include layer 2 (Ethernet) h
                                                                * to rx_callback */
};

int l2_packet_get_own_addr(struct l2_packet_data *l2, u8 *addr)
{
        os_memcpy(addr, l2->own_addr, ETH_ALEN);
        return 0;
}

#ifndef CONFIG_WINPCAP
static int l2_packet_init_libdnet(struct l2_packet_data *l2)
{
        eth_addr_t own_addr;

        l2->eth = eth_open(l2->ifname);
        if (!l2->eth) {
                wpa_printf(MSG_ERROR, "Failed to open interface '%s' - eth_open: %s", l2->ifname,
                return -1;
        }

        if (eth_get(l2->eth, &own_addr) < 0) {
                wpa_printf(MSG_ERROR, "Failed to get own hw address from interface '%s' - eth_get:
                eth_close(l2->eth);
                l2->eth = NULL;
                return -1;
        }
        os_memcpy(l2->own_addr, own_addr.data, ETH_ALEN);

        return 0;
}
#endif                                                  /* CONFIG_WINPCAP */

int l2_packet_send(struct l2_packet_data *l2, const u8 *dst_addr, u16 proto, const u8 *buf, size_t
{
        int ret;
        struct l2_ethhdr *eth;

        if (l2 == NULL) {
                return -1;
        }

        if (l2->l2_hdr) {
#ifdef CONFIG_WINPCAP
```

```
                ret = pcap_sendpacket(l2->pcap, buf, len);
#else                                                   /* CONFIG_WINPCAP */
                ret = eth_send(l2->eth, buf, len);
#endif                                                  /* CONFIG_WINPCAP */
        } else {
                size_t mlen = sizeof(*eth) + len;
                eth = os_malloc(mlen);
                if (eth == NULL) {
                        return -1;
                }

                os_memcpy(eth->h_dest, dst_addr, ETH_ALEN);
                os_memcpy(eth->h_source, l2->own_addr, ETH_ALEN);
                eth->h_proto = htons(proto);
                os_memcpy(eth + 1, buf, len);

#ifdef CONFIG_WINPCAP
                ret = pcap_sendpacket(l2->pcap, (u8 *)eth, mlen);
#else                                                   /* CONFIG_WINPCAP */
                ret = eth_send(l2->eth, (u8 *)eth, mlen);
#endif                                                  /* CONFIG_WINPCAP */

                os_free(eth);
        }

        return ret;
}

#ifndef CONFIG_WINPCAP
static void l2_packet_receive(int sock, void *eloop_ctx, void *sock_ctx)
{
        struct l2_packet_data *l2 = eloop_ctx;
        pcap_t *pcap = sock_ctx;
        struct pcap_pkthdr hdr;
        const u_char *packet;
        struct l2_ethhdr *ethhdr;
        unsigned char *buf;
        size_t len;

        packet = pcap_next(pcap, &hdr);

        if (packet == NULL || hdr.caplen < sizeof(*ethhdr)) {
                return;
        }

        ethhdr = (struct l2_ethhdr *)packet;
        if (l2->l2_hdr) {
                buf = (unsigned char *)ethhdr;
                len = hdr.caplen;
```

```
128         } else {
129                 buf = (unsigned char *)(ethhdr + 1);
130                 len = hdr.caplen - sizeof(*ethhdr);
131         }
132         l2->rx_callback(l2->rx_callback_ctx, ethhdr->h_source, buf, len);
133 }
134 #endif                                                  /* CONFIG_WINPCAP */
135
136 #ifdef CONFIG_WINPCAP
137 static void l2_packet_receive_cb(u_char *user, const struct pcap_pkthdr *hdr, const u_char *pkt_da
138 {
139         struct l2_packet_data *l2 = (struct l2_packet_data *)user;
140         struct l2_ethhdr *ethhdr;
141         unsigned char *buf;
142         size_t len;
143
144         if (pkt_data == NULL || hdr->caplen < sizeof(*ethhdr)) {
145                 return;
146         }
147
148         ethhdr = (struct l2_ethhdr *)pkt_data;
149         if (l2->l2_hdr) {
150                 buf = (unsigned char *)ethhdr;
151                 len = hdr->caplen;
152         } else {
153                 buf = (unsigned char *)(ethhdr + 1);
154                 len = hdr->caplen - sizeof(*ethhdr);
155         }
156         l2->rx_callback(l2->rx_callback_ctx, ethhdr->h_source, buf, len);
157         /*
158          * Use shorter poll interval for 3 seconds to reduce latency during key
159          * handshake.
160          */
161         l2->num_fast_poll = 3 * 50;
162 }
163
164 static void l2_packet_receive_timeout(void *eloop_ctx, void *timeout_ctx)
165 {
166         struct l2_packet_data *l2 = eloop_ctx;
167         pcap_t *pcap = timeout_ctx;
168         int timeout;
169
170         if (l2->num_fast_poll > 0) {
171                 timeout = 20000;
172                 l2->num_fast_poll--;
173         } else {
174                 timeout = 100000;
175         }
176
```

```
177         /* Register new timeout before calling l2_packet_receive() since
178          * receive handler may free this l2_packet instance (which will
179          * cancel this timeout). */
180         eloop_register_timeout(0, timeout, l2_packet_receive_timeout, l2, pcap);
181         pcap_dispatch(pcap, 10, l2_packet_receive_cb, (u_char *)l2);
182 }
183 #endif                                                  /* CONFIG_WINPCAP */
184
185 static int l2_packet_init_libpcap(struct l2_packet_data *l2, unsigned short protocol)
186 {
187         bpf_u_int32 pcap_maskp, pcap_netp;
188         char pcap_filter[200], pcap_err[PCAP_ERRBUF_SIZE];
189         struct bpf_program pcap_fp;
190
191 #ifdef CONFIG_WINPCAP
192         char ifname[128];
193         os_snprintf(ifname, sizeof(ifname), "\\Device\\NPF_%s", l2->ifname);
194         pcap_lookupnet(ifname, &pcap_netp, &pcap_maskp, pcap_err);
195         l2->pcap = pcap_open_live(ifname, 2500, 0, 10, pcap_err);
196         if (l2->pcap == NULL) {
197                 fprintf(stderr, "pcap_open_live: %s\n", pcap_err);
198                 fprintf(stderr, "ifname='%s'\n", ifname);
199                 return -1;
200         }
201         if (pcap_setnonblock(l2->pcap, 1, pcap_err) < 0) {
202                 fprintf(stderr, "pcap_setnonblock: %s\n", pcap_geterr(l2->pcap));
203         }
204 #else                                                   /* CONFIG_WINPCAP */
205         pcap_lookupnet(l2->ifname, &pcap_netp, &pcap_maskp, pcap_err);
206         l2->pcap = pcap_open_live(l2->ifname, 2500, 0, 10, pcap_err);
207         if (l2->pcap == NULL) {
208                 fprintf(stderr, "pcap_open_live: %s\n", pcap_err);
209                 fprintf(stderr, "ifname='%s'\n", l2->ifname);
210                 return -1;
211         }
212         if (pcap_datalink(l2->pcap) != DLT_EN10MB && pcap_set_datalink(l2->pcap, DLT_EN10MB) < 0)
213                 fprintf(stderr, "pcap_set_datalink(DLT_EN10MB): %s\n", pcap_geterr(l2->pcap));
214                 return -1;
215         }
216 #endif                                                  /* CONFIG_WINPCAP */
217         os_snprintf(pcap_filter, sizeof(pcap_filter), "not ether src " MACSTR " and " "( ether dst
218                                 MAC2STR(l2->own_addr), MAC2STR(pae_group_addr), protocol);
219         if (pcap_compile(l2->pcap, &pcap_fp, pcap_filter, 1, pcap_netp) < 0) {
220                 fprintf(stderr, "pcap_compile: %s\n", pcap_geterr(l2->pcap));
221                 return -1;
222         }
223
224         if (pcap_setfilter(l2->pcap, &pcap_fp) < 0) {
225                 fprintf(stderr, "pcap_setfilter: %s\n", pcap_geterr(l2->pcap));
```

```
226                 return -1;
227         }
228
229         pcap_freecode(&pcap_fp);
230 #ifdef BIOCIMMEDIATE
231         /*
232          * When libpcap uses BPF we must enable "immediate mode" to
233          * receive frames right away; otherwise the system may
234          * buffer them for us.
235          */
236         {
237                 unsigned int on = 1;
238                 if (ioctl(pcap_fileno(l2->pcap), BIOCIMMEDIATE, &on) < 0) {
239                         fprintf(stderr, "%s: cannot enable immediate mode on " "interface %s: %s\n
240                         /* XXX should we fail? */
241                 }
242         }
243 #endif                                                  /* BIOCIMMEDIATE */
244
245 #ifdef CONFIG_WINPCAP
246         eloop_register_timeout(0, 100000, l2_packet_receive_timeout, l2, l2->pcap);
247 #else                                                   /* CONFIG_WINPCAP */
248         eloop_register_read_sock(pcap_get_selectable_fd(l2->pcap), l2_packet_receive, l2, l2->pcap
249 #endif                                                  /* CONFIG_WINPCAP */
250
251         return 0;
252 }
253
254 struct l2_packet_data *l2_packet_init(const char *ifname, const u8 *own_addr, unsigned short proto
255 {
256         struct l2_packet_data *l2;
257
258         l2 = os_zalloc(sizeof(struct l2_packet_data));
259         if (l2 == NULL) {
260                 return NULL;
261         }
262         os_strlcpy(l2->ifname, ifname, sizeof(l2->ifname));
263         l2->rx_callback = rx_callback;
264         l2->rx_callback_ctx = rx_callback_ctx;
265         l2->l2_hdr = l2_hdr;
266
267 #ifdef CONFIG_WINPCAP
268         if (own_addr) {
269                 os_memcpy(l2->own_addr, own_addr, ETH_ALEN);
270         }
271 #else                                                   /* CONFIG_WINPCAP */
272         if (l2_packet_init_libdnet(l2)) {
273                 return NULL;
274         }
```

```
#endif                                                  /* CONFIG_WINPCAP */

        if (l2_packet_init_libpcap(l2, protocol)) {
#ifndef CONFIG_WINPCAP
                eth_close(l2->eth);
#endif                                                  /* CONFIG_WINPCAP */
                os_free(l2);
                return NULL;
        }

        return l2;
}

void l2_packet_deinit(struct l2_packet_data *l2)
{
        if (l2 == NULL) {
                return;
        }

#ifdef CONFIG_WINPCAP
        eloop_cancel_timeout(l2_packet_receive_timeout, l2, l2->pcap);
#else                                                   /* CONFIG_WINPCAP */
        if (l2->eth) {
                eth_close(l2->eth);
        }
        eloop_unregister_read_sock(pcap_get_selectable_fd(l2->pcap));
#endif                                                  /* CONFIG_WINPCAP */
        if (l2->pcap) {
                pcap_close(l2->pcap);
        }
        os_free(l2);
}

int l2_packet_get_ip_addr(struct l2_packet_data *l2, char *buf, size_t len)
{
        pcap_if_t *devs, *dev;
        struct pcap_addr *addr;
        struct sockaddr_in *saddr;
        int found = 0;
        char err[PCAP_ERRBUF_SIZE + 1];

        if (pcap_findalldevs(&devs, err) < 0) {
                wpa_printf(MSG_DEBUG, "pcap_findalldevs: %s\n", err);
                return -1;
        }

        for (dev = devs; dev && !found; dev = dev->next) {
                if (os_strcmp(dev->name, l2->ifname) != 0) {
                        continue;
```

```
                }

                addr = dev->addresses;
                while (addr) {
                        saddr = (struct sockaddr_in *)addr->addr;
                        if (saddr && saddr->sin_family == AF_INET) {
                                os_strlcpy(buf, inet_ntoa(saddr->sin_addr), len);
                                found = 1;
                                break;
                        }
                        addr = addr->next;
                }
        }

        pcap_freealldevs(devs);

        return found ? 0 : -1;
}

void l2_packet_notify_auth_start(struct l2_packet_data *l2)
{
#ifdef CONFIG_WINPCAP
        /*
         * Use shorter poll interval for 3 seconds to reduce latency during key
         * handshake.
         */
        l2->num_fast_poll = 3 * 50;
        eloop_cancel_timeout(l2_packet_receive_timeout, l2, l2->pcap);
        eloop_register_timeout(0, 10000, l2_packet_receive_timeout, l2, l2->pcap);
#endif                                                  /* CONFIG_WINPCAP */
}

int l2_packet_set_packet_filter(struct l2_packet_data *l2, enum l2_packet_filter_type type)
{
        return -1;
}
```