

static const u8 pae_group_addr[ETH_ALEN] = { 0x01, 0x80, 0xc2, 0x00, 0x00, 0x03 };

16 17

18 19

20

21 22

23 24

25

26 27

28

29

#else

#include "common.h"

#include "eloop.h"

#include "12_packet.h"

struct 12_packet_data {

#ifdef CONFIG_WINPCAP

pcap_t *pcap;

eth_t *eth;

unsigned int num_fast_poll;

/* CONFIG WINPCAP */

/* CONFIG WINPCAP */

```
30
     #endif
                                                                /* CONFIG_WINPCAP */
31
              char ifname[100];
32
             u8 own addr[ETH ALEN];
             void (*rx_callback)(void *ctx, const u8 *src_addr, const u8 *buf, size_t len);
33
             void *rx_callback_ctx;
34
35
             int 12 hdr;
                                                               /* whether to include layer 2 (Ethernet) h
                                                                         * to rx callback */
36
37
     };
38
39
     int 12_packet_get_own_addr(struct 12_packet_data *12, u8 *addr)
40
41
              os_memcpy(addr, 12->own_addr, ETH_ALEN);
42
             return 0;
43
     }
44
45
     #ifndef CONFIG_WINPCAP
46
     static int 12_packet_init_libdnet(struct 12_packet_data *12)
47
48
              eth_addr_t own_addr;
49
50
             12->eth = eth_open(12->ifname);
51
              if (!12->eth) {
52
                      wpa_printf(MSG_ERROR, "Failed to open interface '%s' - eth_open: %s", 12->ifname,
53
                      return -1;
54
              }
55
56
             if (eth_get(12->eth, &own_addr) < 0) {</pre>
57
                      wpa_printf(MSG_ERROR, "Failed to get own hw address from interface '%s' - eth_get:
58
                      eth_close(12->eth);
59
                      12->eth = NULL;
60
                      return -1;
61
              }
62
             os_memcpy(12->own_addr, own_addr.data, ETH_ALEN);
63
64
             return 0;
65
66
     #endif
                                                               /* CONFIG_WINPCAP */
67
68
     int 12_packet_send(struct 12_packet_data *12, const u8 *dst_addr, u16 proto, const u8 *buf, size_t
69
70
              int ret;
71
             struct 12_ethhdr *eth;
72
73
             if (12 == NULL) {
74
                      return -1;
75
              }
76
77
              if (12->12_hdr) {
78
     #ifdef CONFIG_WINPCAP
```

```
79
                       ret = pcap_sendpacket(12->pcap, buf, len);
80
      #else
                                                                 /* CONFIG WINPCAP */
81
                       ret = eth send(12->eth, buf, len);
      #endif
                                                                 /* CONFIG_WINPCAP */
82
              } else {
83
84
                       size t mlen = sizeof(*eth) + len;
85
                       eth = os_malloc(mlen);
                       if (eth == NULL) {
86
87
                               return -1;
88
                       }
89
90
                       os_memcpy(eth->h_dest, dst_addr, ETH_ALEN);
91
                       os_memcpy(eth->h_source, 12->own_addr, ETH_ALEN);
92
                       eth->h_proto = htons(proto);
93
                       os_memcpy(eth + 1, buf, len);
94
95
      #ifdef CONFIG_WINPCAP
96
                       ret = pcap sendpacket(12->pcap, (u8 *)eth, mlen);
97
      #else
                                                                 /* CONFIG_WINPCAP */
98
                       ret = eth_send(12->eth, (u8 *)eth, mlen);
99
      #endif
                                                                 /* CONFIG WINPCAP */
100
                       os_free(eth);
101
102
              }
103
104
              return ret;
105
      }
106
107
      #ifndef CONFIG_WINPCAP
108
      static void 12_packet_receive(int sock, void *eloop_ctx, void *sock_ctx)
109
      {
              struct 12_packet_data *12 = eloop_ctx;
110
              pcap_t *pcap = sock_ctx;
111
              struct pcap_pkthdr hdr;
112
113
              const u_char *packet;
              struct 12_ethhdr *ethhdr;
114
115
              unsigned char *buf;
116
              size_t len;
117
118
              packet = pcap_next(pcap, &hdr);
119
120
              if (packet == NULL || hdr.caplen < sizeof(*ethhdr)) {</pre>
121
                       return;
122
              }
123
124
              ethhdr = (struct 12_ethhdr *)packet;
125
               if (12->12_hdr) {
                       buf = (unsigned char *)ethhdr;
126
                       len = hdr.caplen;
127
```

```
} else {
128
129
                       buf = (unsigned char *)(ethhdr + 1);
130
                       len = hdr.caplen - sizeof(*ethhdr);
131
               }
              12->rx_callback(12->rx_callback_ctx, ethhdr->h_source, buf, len);
132
133
      #endif
                                                                /* CONFIG WINPCAP */
134
135
136
      #ifdef CONFIG WINPCAP
137
      static void 12_packet_receive_cb(u_char *user, const struct pcap_pkthdr *hdr, const u_char *pkt_da
138
139
               struct 12_packet_data *12 = (struct 12_packet_data *)user;
140
               struct 12_ethhdr *ethhdr;
141
              unsigned char *buf;
142
              size_t len;
143
144
              if (pkt_data == NULL || hdr->caplen < sizeof(*ethhdr)) {</pre>
145
                       return;
146
              }
147
148
              ethhdr = (struct 12_ethhdr *)pkt_data;
149
              if (12->12 hdr) {
                       buf = (unsigned char *)ethhdr;
150
151
                       len = hdr->caplen;
152
               } else {
153
                       buf = (unsigned char *)(ethhdr + 1);
154
                       len = hdr->caplen - sizeof(*ethhdr);
155
               }
              12->rx_callback(12->rx_callback_ctx, ethhdr->h_source, buf, len);
156
157
158
               * Use shorter poll interval for 3 seconds to reduce latency during key
               * handshake.
159
160
               */
161
              12->num_fast_poll = 3 * 50;
162
163
164
      static void 12_packet_receive_timeout(void *eloop_ctx, void *timeout_ctx)
165
      {
166
              struct 12_packet_data *12 = eloop_ctx;
167
              pcap_t *pcap = timeout_ctx;
168
              int timeout;
169
170
              if (12->num_fast_poll > 0) {
171
                       timeout = 20000;
                       12->num_fast_poll--;
172
173
               } else {
174
                       timeout = 100000;
175
               }
176
```

```
/* Register new timeout before calling 12_packet_receive() since
177
178
                * receive handler may free this 12 packet instance (which will
179
                * cancel this timeout). */
180
              eloop_register_timeout(0, timeout, 12_packet_receive_timeout, 12, pcap);
181
              pcap_dispatch(pcap, 10, 12_packet_receive_cb, (u_char *)12);
182
      #endif
183
                                                                /* CONFIG WINPCAP */
184
185
      static int 12 packet init libpcap(struct 12 packet data *12, 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", 12->ifname);
194
              pcap lookupnet(ifname, &pcap netp, &pcap maskp, pcap err);
195
              12->pcap = pcap_open_live(ifname, 2500, 0, 10, pcap_err);
196
              if (12->pcap == NULL) {
197
                       fprintf(stderr, "pcap_open_live: %s\n", pcap_err);
                       fprintf(stderr, "ifname='%s'\n", ifname);
198
199
                       return -1;
200
              }
201
              if (pcap setnonblock(12->pcap, 1, pcap err) < 0) {</pre>
202
                       fprintf(stderr, "pcap_setnonblock: %s\n", pcap_geterr(12->pcap));
203
              }
204
      #else
                                                                /* CONFIG WINPCAP */
              pcap_lookupnet(12->ifname, &pcap_netp, &pcap_maskp, pcap_err);
205
206
              12->pcap = pcap_open_live(l2->ifname, 2500, 0, 10, pcap_err);
207
              if (12->pcap == NULL) {
                       fprintf(stderr, "pcap_open_live: %s\n", pcap_err);
208
209
                       fprintf(stderr, "ifname='%s'\n", 12->ifname);
210
                       return -1;
211
              }
212
              if (pcap_datalink(12->pcap) != DLT_EN10MB && pcap_set_datalink(12->pcap, DLT_EN10MB) < 0)</pre>
213
                       fprintf(stderr, "pcap_set_datalink(DLT_EN10MB): %s\n", pcap_geterr(12->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(12->own_addr), MAC2STR(pae_group_addr), protocol);
219
              if (pcap_compile(12->pcap, &pcap_fp, pcap_filter, 1, pcap_netp) < 0) {</pre>
220
                       fprintf(stderr, "pcap compile: %s\n", pcap geterr(12->pcap));
221
                       return -1;
222
              }
223
224
              if (pcap setfilter(12->pcap, &pcap fp) < 0) {</pre>
225
                       fprintf(stderr, "pcap_setfilter: %s\n", pcap_geterr(12->pcap));
```

```
226
                       return -1;
227
              }
228
229
              pcap_freecode(&pcap_fp);
      #ifdef BIOCIMMEDIATE
230
231
               * When libpcap uses BPF we must enable "immediate mode" to
232
               * receive frames right away; otherwise the system may
233
234
               * buffer them for us.
               */
235
236
               {
237
                       unsigned int on = 1;
238
                       if (ioctl(pcap_fileno(12->pcap), BIOCIMMEDIATE, &on) < 0) {</pre>
                               fprintf(stderr, "%s: cannot enable immediate mode on " "interface %s: %s\n
239
                               /* XXX should we fail? */
240
                       }
241
242
              }
243
      #endif
                                                                 /* BIOCIMMEDIATE */
244
245
      #ifdef CONFIG_WINPCAP
246
               eloop_register_timeout(0, 100000, 12_packet_receive_timeout, 12, 12->pcap);
                                                                /* CONFIG_WINPCAP */
247
      #else
248
              eloop_register_read_sock(pcap_get_selectable_fd(12->pcap), 12_packet_receive, 12, 12->pcap
249
      #endif
                                                                /* CONFIG_WINPCAP */
250
251
              return 0;
252
      }
253
254
      struct 12_packet_data *12_packet_init(const char *ifname, const u8 *own_addr, unsigned short proto
255
      {
256
              struct 12_packet_data *12;
257
258
              12 = os_zalloc(sizeof(struct 12_packet_data));
259
              if (12 == NULL) {
260
                       return NULL;
261
              }
262
              os_strlcpy(12->ifname, ifname, sizeof(12->ifname));
263
              12->rx_callback = rx_callback;
264
              12->rx_callback_ctx = rx_callback_ctx;
265
              12->12_hdr = 12_hdr;
266
267
      #ifdef CONFIG_WINPCAP
268
              if (own_addr) {
269
                       os memcpy(12->own addr, own addr, ETH ALEN);
270
              }
271
      #else
                                                                 /* CONFIG_WINPCAP */
272
              if (12_packet_init_libdnet(12)) {
273
                       return NULL;
              }
274
```

```
275
      #endif
                                                                 /* CONFIG_WINPCAP */
276
277
               if (12 packet init libpcap(12, protocol)) {
      #ifndef CONFIG_WINPCAP
278
279
                       eth_close(12->eth);
280
      #endif
                                                                 /* CONFIG WINPCAP */
281
                       os free(12);
282
                       return NULL;
283
               }
284
285
              return 12;
286
      }
287
288
      void 12_packet_deinit(struct 12_packet_data *12)
289
290
              if (12 == NULL) {
291
                       return;
292
               }
293
294
      #ifdef CONFIG_WINPCAP
295
               eloop_cancel_timeout(12_packet_receive_timeout, 12, 12->pcap);
                                                                 /* CONFIG WINPCAP */
296
      #else
              if (12->eth) {
297
298
                       eth_close(12->eth);
299
               }
300
               eloop_unregister_read_sock(pcap_get_selectable_fd(12->pcap));
      #endif
                                                                 /* CONFIG_WINPCAP */
301
302
               if (12->pcap) {
303
                       pcap_close(12->pcap);
304
               }
305
              os_free(12);
306
      }
307
308
      int 12_packet_get_ip_addr(struct 12_packet_data *12, char *buf, size_t len)
309
310
              pcap_if_t *devs, *dev;
311
               struct pcap_addr *addr;
312
               struct sockaddr_in *saddr;
313
              int found = 0;
314
               char err[PCAP_ERRBUF_SIZE + 1];
315
316
               if (pcap_findalldevs(&devs, err) < 0) {</pre>
317
                       wpa_printf(MSG_DEBUG, "pcap_findalldevs: %s\n", err);
318
                       return -1;
319
               }
320
321
               for (dev = devs; dev && !found; dev = dev->next) {
                       if (os_strcmp(dev->name, 12->ifname) != 0) {
322
323
                               continue;
```

```
324
                      }
325
                       addr = dev->addresses;
326
                      while (addr) {
327
                               saddr = (struct sockaddr_in *)addr->addr;
328
                               if (saddr && saddr->sin_family == AF_INET) {
329
                                       os_strlcpy(buf, inet_ntoa(saddr->sin_addr), len);
330
                                       found = 1;
331
332
                                       break;
                               }
333
                               addr = addr->next;
334
335
                      }
336
              }
337
338
              pcap_freealldevs(devs);
339
              return found ? 0 : -1;
340
341
      }
342
343
      void 12_packet_notify_auth_start(struct 12_packet_data *12)
344
      #ifdef CONFIG WINPCAP
345
              /*
346
347
               * Use shorter poll interval for 3 seconds to reduce latency during key
348
               * handshake.
               */
349
              12->num_fast_poll = 3 * 50;
350
351
              eloop_cancel_timeout(12_packet_receive_timeout, 12, 12->pcap);
              eloop_register_timeout(0, 10000, 12_packet_receive_timeout, 12, 12->pcap);
352
      #endif
                                                                /* CONFIG_WINPCAP */
353
354
      }
355
      int 12_packet_set_packet_filter(struct 12_packet_data *12, enum 12_packet_filter_type type)
356
357
      {
358
              return -1;
359
      }
```