Linux Cross Reference

Free Electrons

Embedded Linux Experts

• *source navigation* • diff markup • identifier search • freetext search •

Version: 2.0.40 2.2.26 2.4.37 3.6 3.7 3.8 3.9 3.10 3.11 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 ***4.2***

Linux/net/ipv4/ip_options.c

```
 1 /*
 2  * INET         An implementation of the TCP/IP protocol suite for the LINUX
 3  *              operating system.  INET is implemented using the  BSD Socket
 4  *              interface as the means of communication with the user level.
 5  *
 6  *              The options processing module for ip.c
 7  *
 8  * Authors:     A.N.Kuznetsov
 9  *
10  */
11
12 #define pr_fmt(fmt) "IPv4: " fmt
13
14 #include <linux/capability.h>
15 #include <linux/module.h>
16 #include <linux/slab.h>
17 #include <linux/types.h>
18 #include <asm/uaccess.h>
19 #include <asm/unaligned.h>
20 #include <linux/skbuff.h>
21 #include <linux/ip.h>
22 #include <linux/icmp.h>
23 #include <linux/netdevice.h>
24 #include <linux/rtnetlink.h>
25 #include <net/sock.h>
26 #include <net/ip.h>
27 #include <net/icmp.h>
28 #include <net/route.h>
29 #include <net/cipso_ipv4.h>
30 #include <net/ip_fib.h>
31
32 /*
33  * Write options to IP header, record destination address to
34  * source route option, address of outgoing interface
35  * (we should already know it, so that this  function is allowed be
36  * called only after routing decision) and timestamp,
37  * if we originate this datagram.
38  *
39  * daddr is real destination address, next hop is recorded in IP header.
40  * saddr is address of outgoing interface.
41  */
42
43 void ip_options_build(struct sk_buff *skb, struct ip_options *opt,
44                       __be32 daddr, struct rtable *rt, int is_frag)
45 {
46         unsigned char *iph = skb_network_header(skb);
47
48         memcpy(&(IPCB(skb)->opt), opt, sizeof(struct ip_options));
49         memcpy(iph+sizeof(struct iphdr), opt->__data, opt->optlen);
50         opt = &(IPCB(skb)->opt);
51
52         if (opt->srr)
53                 memcpy(iph+opt->srr+iph[opt->srr+1]-4, &daddr, 4);
54
55         if (!is_frag) {
56                 if (opt->rr_needaddr)
57                         ip_rt_get_source(iph+opt->rr+iph[opt->rr+2]-5, skb, rt);
58                 if (opt->ts_needaddr)
59                         ip_rt_get_source(iph+opt->ts+iph[opt->ts+2]-9, skb, rt);
60                 if (opt->ts_needtime) {
61                         struct timespec tv;
62                         __be32 midtime;
63                         getnstimeofday(&tv);
64                         midtime = htonl((tv.tv_sec % 86400) * MSEC_PER_SEC + tv.tv_nsec / NSEC_PER_MSEC);
65                         memcpy(iph+opt->ts+iph[opt->ts+2]-5, &midtime, 4);
```

```
66                 }
67                 return;
68         }
69         if (opt->rr) {
70                 memset(iph+opt->rr, IPOPT_NOP, iph[opt->rr+1]);
71                 opt->rr = 0;
72                 opt->rr_needaddr = 0;
73         }
74         if (opt->ts) {
75                 memset(iph+opt->ts, IPOPT_NOP, iph[opt->ts+1]);
76                 opt->ts = 0;
77                 opt->ts_needaddr = opt->ts_needtime = 0;
78         }
79 }
80
81 /*
82  * Provided (sopt, skb) points to received options,
83  * build in dopt compiled option set appropriate for answering.
84  * i.e. invert SRR option, copy anothers,
85  * and grab room in RR/TS options.
86  *
87  * NOTE: dopt cannot point to skb.
88  */
89
90 int __ip_options_echo(struct ip_options *dopt, struct sk_buff *skb,
91                       const struct ip_options *sopt)
92 {
93         unsigned char *sptr, *dptr;
94         int soffset, doffset;
95         int     optlen;
96
97         memset(dopt, 0, sizeof(struct ip_options));
98
99         if (sopt->optlen == 0)
100                return 0;
101
102        sptr = skb_network_header(skb);
103        dptr = dopt->__data;
104
105        if (sopt->rr) {
106                optlen  = sptr[sopt->rr+1];
107                soffset = sptr[sopt->rr+2];
108                dopt->rr = dopt->optlen + sizeof(struct iphdr);
109                memcpy(dptr, sptr+sopt->rr, optlen);
110                if (sopt->rr_needaddr && soffset <= optlen) {
111                        if (soffset + 3 > optlen)
112                                return -EINVAL;
113                        dptr[2] = soffset + 4;
114                        dopt->rr_needaddr = 1;
115                }
116                dptr += optlen;
117                dopt->optlen += optlen;
118        }
119        if (sopt->ts) {
120                optlen = sptr[sopt->ts+1];
121                soffset = sptr[sopt->ts+2];
122                dopt->ts = dopt->optlen + sizeof(struct iphdr);
123                memcpy(dptr, sptr+sopt->ts, optlen);
124                if (soffset <= optlen) {
125                        if (sopt->ts_needaddr) {
126                                if (soffset + 3 > optlen)
127                                        return -EINVAL;
128                                dopt->ts_needaddr = 1;
129                                soffset += 4;
130                        }
131                        if (sopt->ts_needtime) {
132                                if (soffset + 3 > optlen)
133                                        return -EINVAL;
134                                if ((dptr[3]&0xF) != IPOPT_TS_PRESPEC) {
135                                        dopt->ts_needtime = 1;
136                                        soffset += 4;
137                                } else {
138                                        dopt->ts_needtime = 0;
139
140                                        if (soffset + 7 <= optlen) {
141                                                __be32 addr;
142
143                                                memcpy(&addr, dptr+soffset-1, 4);
144                                                if (inet_addr_type(dev_net(skb_dst(skb)->dev), addr) != RTN_UNICAST) {
145                                                        dopt->ts_needtime = 1;
146                                                        soffset += 8;
147                                                }
148                                        }
149                                }
150                        }
```

```
151                         dptr[2] = soffset;
152                 }
153                 dptr += optlen;
154                 dopt->optlen += optlen;
155         }
156         if (sopt->srr) {
157                 unsigned char *start = sptr+sopt->srr;
158                 __be32 faddr;
159
160                 optlen  = start[1];
161                 soffset = start[2];
162                 doffset = 0;
163                 if (soffset > optlen)
164                         soffset = optlen + 1;
165                 soffset -= 4;
166                 if (soffset > 3) {
167                         memcpy(&faddr, &start[soffset-1], 4);
168                         for (soffset -= 4, doffset = 4; soffset > 3; soffset -= 4, doffset += 4)
169                                 memcpy(&dptr[doffset-1], &start[soffset-1], 4);
170                         /*
171                          * RFC1812 requires to fix illegal source routes.
172                          */
173                         if (memcmp(&ip_hdr(skb)->saddr,
174                                    &start[soffset + 3], 4) == 0)
175                                 doffset -= 4;
176                 }
177                 if (doffset > 3) {
178                         __be32 daddr = fib_compute_spec_dst(skb);
179
180                         memcpy(&start[doffset-1], &daddr, 4);
181                         dopt->faddr = faddr;
182                         dptr[0] = start[0];
183                         dptr[1] = doffset+3;
184                         dptr[2] = 4;
185                         dptr += doffset+3;
186                         dopt->srr = dopt->optlen + sizeof(struct iphdr);
187                         dopt->optlen += doffset+3;
188                         dopt->is_strictroute = sopt->is_strictroute;
189                 }
190         }
191         if (sopt->cipso) {
192                 optlen  = sptr[sopt->cipso+1];
193                 dopt->cipso = dopt->optlen+sizeof(struct iphdr);
194                 memcpy(dptr, sptr+sopt->cipso, optlen);
195                 dptr += optlen;
196                 dopt->optlen += optlen;
197         }
198         while (dopt->optlen & 3) {
199                 *dptr++ = IPOPT_END;
200                 dopt->optlen++;
201         }
202         return 0;
203 }
204
205 /*
206  *      Options "fragmenting", just fill options not
207  *      allowed in fragments with NOOPs.
208  *      Simple and stupid 8), but the most efficient way.
209  */
210
211 void ip_options_fragment(struct sk_buff *skb)
212 {
213         unsigned char *optptr = skb_network_header(skb) + sizeof(struct iphdr);
214         struct ip_options *opt = &(IPCB(skb)->opt);
215         int  l = opt->optlen;
216         int  optlen;
217
218         while (l > 0) {
219                 switch (*optptr) {
220                 case IPOPT_END:
221                         return;
222                 case IPOPT_NOOP:
223                         l--;
224                         optptr++;
225                         continue;
226                 }
227                 optlen = optptr[1];
228                 if (optlen < 2 || optlen > l)
229                   return;
230                 if (!IPOPT_COPIED(*optptr))
231                         memset(optptr, IPOPT_NOOP, optlen);
232                 l -= optlen;
233                 optptr += optlen;
234         }
235         opt->ts = 0;
```

```
236         opt->rr = 0;
237         opt->rr_needaddr = 0;
238         opt->ts_needaddr = 0;
239         opt->ts_needtime = 0;
240 }
241
242 /* helper used by ip_options_compile() to call fib_compute_spec_dst()
243  * at most one time.
244  */
245 static void spec_dst_fill(__be32 *spec_dst, struct sk_buff *skb)
246 {
247         if (*spec_dst == htonl(INADDR_ANY))
248                 *spec_dst = fib_compute_spec_dst(skb);
249 }
250
251 /*
252  * Verify options and fill pointers in struct options.
253  * Caller should clear *opt, and set opt->data.
254  * If opt == NULL, then skb->data should point to IP header.
255  */
256
257 int ip_options_compile(struct net *net,
258                        struct ip_options *opt, struct sk_buff *skb)
259 {
260         __be32 spec_dst = htonl(INADDR_ANY);
261         unsigned char *pp_ptr = NULL;
262         struct rtable *rt = NULL;
263         unsigned char *optptr;
264         unsigned char *iph;
265         int optlen, l;
266
267         if (skb) {
268                 rt = skb_rtable(skb);
269                 optptr = (unsigned char *)&(ip_hdr(skb)[1]);
270         } else
271                 optptr = opt->__data;
272         iph = optptr - sizeof(struct iphdr);
273
274         for (l = opt->optlen; l > 0; ) {
275                 switch (*optptr) {
276                 case IPOPT_END:
277                         for (optptr++, l--; l > 0; optptr++, l--) {
278                                 if (*optptr != IPOPT_END) {
279                                         *optptr = IPOPT_END;
280                                         opt->is_changed = 1;
281                                 }
282                         }
283                         goto eol;
284                 case IPOPT_NOOP:
285                         l--;
286                         optptr++;
287                         continue;
288                 }
289                 if (unlikely(l < 2)) {
290                         pp_ptr = optptr;
291                         goto error;
292                 }
293                 optlen = optptr[1];
294                 if (optlen < 2 || optlen > l) {
295                         pp_ptr = optptr;
296                         goto error;
297                 }
298                 switch (*optptr) {
299                 case IPOPT_SSRR:
300                 case IPOPT_LSRR:
301                         if (optlen < 3) {
302                                 pp_ptr = optptr + 1;
303                                 goto error;
304                         }
305                         if (optptr[2] < 4) {
306                                 pp_ptr = optptr + 2;
307                                 goto error;
308                         }
309                         /* NB: cf RFC-1812 5.2.4.1 */
310                         if (opt->srr) {
311                                 pp_ptr = optptr;
312                                 goto error;
313                         }
314                         if (!skb) {
315                                 if (optptr[2] != 4 || optlen < 7 || ((optlen-3) & 3)) {
316                                         pp_ptr = optptr + 1;
317                                         goto error;
318                                 }
319                                 memcpy(&opt->faddr, &optptr[3], 4);
320                                 if (optlen > 7)
```

```
321                                     memmove(&optptr[3], &optptr[7], optlen-7);
322                             }
323                             opt->is_strictroute = (optptr[0] == IPOPT_SSRR);
324                             opt->srr = optptr - iph;
325                             break;
326                     case IPOPT_RR:
327                             if (opt->rr) {
328                                     pp_ptr = optptr;
329                                     goto error;
330                             }
331                             if (optlen < 3) {
332                                     pp_ptr = optptr + 1;
333                                     goto error;
334                             }
335                             if (optptr[2] < 4) {
336                                     pp_ptr = optptr + 2;
337                                     goto error;
338                             }
339                             if (optptr[2] <= optlen) {
340                                     if (optptr[2]+3 > optlen) {
341                                             pp_ptr = optptr + 2;
342                                             goto error;
343                                     }
344                                     if (rt) {
345                                             spec_dst_fill(&spec_dst, skb);
346                                             memcpy(&optptr[optptr[2]-1], &spec_dst, 4);
347                                             opt->is_changed = 1;
348                                     }
349                                     optptr[2] += 4;
350                                     opt->rr_needaddr = 1;
351                             }
352                             opt->rr = optptr - iph;
353                             break;
354                     case IPOPT_TIMESTAMP:
355                             if (opt->ts) {
356                                     pp_ptr = optptr;
357                                     goto error;
358                             }
359                             if (optlen < 4) {
360                                     pp_ptr = optptr + 1;
361                                     goto error;
362                             }
363                             if (optptr[2] < 5) {
364                                     pp_ptr = optptr + 2;
365                                     goto error;
366                             }
367                             if (optptr[2] <= optlen) {
368                                     unsigned char *timeptr = NULL;
369                                     if (optptr[2]+3 > optlen) {
370                                             pp_ptr = optptr + 2;
371                                             goto error;
372                                     }
373                                     switch (optptr[3]&0xF) {
374                                     case IPOPT_TS_TSONLY:
375                                             if (skb)
376                                                     timeptr = &optptr[optptr[2]-1];
377                                             opt->ts_needtime = 1;
378                                             optptr[2] += 4;
379                                             break;
380                                     case IPOPT_TS_TSANDADDR:
381                                             if (optptr[2]+7 > optlen) {
382                                                     pp_ptr = optptr + 2;
383                                                     goto error;
384                                             }
385                                             if (rt)  {
386                                                     spec_dst_fill(&spec_dst, skb);
387                                                     memcpy(&optptr[optptr[2]-1], &spec_dst, 4);
388                                                     timeptr = &optptr[optptr[2]+3];
389                                             }
390                                             opt->ts_needaddr = 1;
391                                             opt->ts_needtime = 1;
392                                             optptr[2] += 8;
393                                             break;
394                                     case IPOPT_TS_PRESPEC:
395                                             if (optptr[2]+7 > optlen) {
396                                                     pp_ptr = optptr + 2;
397                                                     goto error;
398                                             }
399                                             {
400                                                     __be32 addr;
401                                                     memcpy(&addr, &optptr[optptr[2]-1], 4);
402                                                     if (inet_addr_type(net, addr) == RTN_UNICAST)
403                                                             break;
404                                                     if (skb)
405                                                             timeptr = &optptr[optptr[2]+3];
```

```
406                                         }
407                                         opt->ts_needtime = 1;
408                                         optptr[2] += 8;
409                                         break;
410                                 default:
411                                         if (!skb && !ns_capable(net->user_ns, CAP_NET_RAW)) {
412                                                 pp_ptr = optptr + 3;
413                                                 goto error;
414                                         }
415                                         break;
416                                 }
417                                 if (timeptr) {
418                                         struct timespec tv;
419                                         u32  midtime;
420                                         getnstimeofday(&tv);
421                                         midtime = (tv.tv_sec % 86400) * MSEC_PER_SEC + tv.tv_nsec / NSEC_PER_MSEC;
422                                         put_unaligned_be32(midtime, timeptr);
423                                         opt->is_changed = 1;
424                                 }
425                         } else if ((optptr[3]&0xF) != IPOPT_TS_PRESPEC) {
426                                 unsigned int overflow = optptr[3]>>4;
427                                 if (overflow == 15) {
428                                         pp_ptr = optptr + 3;
429                                         goto error;
430                                 }
431                                 if (skb) {
432                                         optptr[3] = (optptr[3]&0xF)|((overflow+1)<<4);
433                                         opt->is_changed = 1;
434                                 }
435                         }
436                         opt->ts = optptr - iph;
437                         break;
438                 case IPOPT_RA:
439                         if (optlen < 4) {
440                                 pp_ptr = optptr + 1;
441                                 goto error;
442                         }
443                         if (optptr[2] == 0 && optptr[3] == 0)
444                                 opt->router_alert = optptr - iph;
445                         break;
446                 case IPOPT_CIPSO:
447                         if ((!skb && !ns_capable(net->user_ns, CAP_NET_RAW)) || opt->cipso) {
448                                 pp_ptr = optptr;
449                                 goto error;
450                         }
451                         opt->cipso = optptr - iph;
452                         if (cipso_v4_validate(skb, &optptr)) {
453                                 pp_ptr = optptr;
454                                 goto error;
455                         }
456                         break;
457                 case IPOPT_SEC:
458                 case IPOPT_SID:
459                 default:
460                         if (!skb && !ns_capable(net->user_ns, CAP_NET_RAW)) {
461                                 pp_ptr = optptr;
462                                 goto error;
463                         }
464                         break;
465                 }
466                 l -= optlen;
467                 optptr += optlen;
468         }
469
470 eol:
471         if (!pp_ptr)
472                 return 0;
473
474 error:
475         if (skb) {
476                 icmp_send(skb, ICMP_PARAMETERPROB, 0, htonl((pp_ptr-iph)<<24));
477         }
478         return -EINVAL;
479 }
480 EXPORT_SYMBOL(ip_options_compile);
481
482 /*
483  *      Undo all the changes done by ip_options_compile().
484  */
485
486 void ip_options_undo(struct ip_options *opt)
487 {
488         if (opt->srr) {
489                 unsigned  char *optptr = opt->__data+opt->srr-sizeof(struct  iphdr);
490                 memmove(optptr+7, optptr+3, optptr[1]-7);
```

```
491                 memcpy(optptr+3, &opt->faddr, 4);
492         }
493         if (opt->rr_needaddr) {
494                 unsigned  char *optptr = opt->__data+opt->rr-sizeof(struct  iphdr);
495                 optptr[2] -= 4;
496                 memset(&optptr[optptr[2]-1], 0, 4);
497         }
498         if (opt->ts) {
499                 unsigned  char *optptr = opt->__data+opt->ts-sizeof(struct  iphdr);
500                 if (opt->ts_needtime) {
501                         optptr[2] -= 4;
502                         memset(&optptr[optptr[2]-1], 0, 4);
503                         if ((optptr[3]&0xF) == IPOPT_TS_PRESPEC)
504                                 optptr[2] -= 4;
505                 }
506                 if (opt->ts_needaddr) {
507                         optptr[2] -= 4;
508                         memset(&optptr[optptr[2]-1], 0, 4);
509                 }
510         }
511 }
512
513 static struct ip_options_rcu *ip_options_get_alloc(const int optlen)
514 {
515         return kzalloc(sizeof(struct ip_options_rcu) + ((optlen + 3) & ~3),
516                        GFP_KERNEL);
517 }
518
519 static int ip_options_get_finish(struct net *net, struct ip_options_rcu **optp,
520                                  struct ip_options_rcu *opt, int optlen)
521 {
522         while (optlen & 3)
523                 opt->opt.__data[optlen++] = IPOPT_END;
524         opt->opt.optlen = optlen;
525         if (optlen && ip_options_compile(net, &opt->opt, NULL)) {
526                 kfree(opt);
527                 return -EINVAL;
528         }
529         kfree(*optp);
530         *optp = opt;
531         return 0;
532 }
533
534 int ip_options_get_from_user(struct net *net, struct ip_options_rcu **optp,
535                              unsigned char __user *data, int optlen)
536 {
537         struct ip_options_rcu *opt = ip_options_get_alloc(optlen);
538
539         if (!opt)
540                 return -ENOMEM;
541         if (optlen && copy_from_user(opt->opt.__data, data, optlen)) {
542                 kfree(opt);
543                 return -EFAULT;
544         }
545         return ip_options_get_finish(net, optp, opt, optlen);
546 }
547
548 int ip_options_get(struct net *net, struct ip_options_rcu **optp,
549                    unsigned char *data, int optlen)
550 {
551         struct ip_options_rcu *opt = ip_options_get_alloc(optlen);
552
553         if (!opt)
554                 return -ENOMEM;
555         if (optlen)
556                 memcpy(opt->opt.__data, data, optlen);
557         return ip_options_get_finish(net, optp, opt, optlen);
558 }
559
560 void ip_forward_options(struct sk_buff *skb)
561 {
562         struct   ip_options *opt        = &(IPCB(skb)->opt);
563         unsigned char *optptr;
564         struct rtable *rt = skb_rtable(skb);
565         unsigned char *raw = skb_network_header(skb);
566
567         if (opt->rr_needaddr) {
568                 optptr = (unsigned char *)raw + opt->rr;
569                 ip_rt_get_source(&optptr[optptr[2]-5], skb, rt);
570                 opt->is_changed = 1;
571         }
572         if (opt->srr_is_hit) {
573                 int srrptr, srrspace;
574
575                 optptr = raw + opt->srr;
```

```
576
577                 for ( srrptr = optptr[2], srrspace = optptr[1];
578                      srrptr <= srrspace;
579                      srrptr += 4
580                      ) {
581                         if (srrptr + 3 > srrspace)
582                                 break;
583                         if (memcmp(&opt->nexthop, &optptr[srrptr-1], 4) == 0)
584                                 break;
585                 }
586                 if (srrptr + 3 <= srrspace) {
587                         opt->is_changed = 1;
588                         ip_hdr(skb)->daddr = opt->nexthop;
589                         ip_rt_get_source(&optptr[srrptr-1], skb, rt);
590                         optptr[2] = srrptr+4;
591                 } else {
592                         net_crit_ratelimited("%s(): Argh! Destination lost!\n",
593                                              __func__);
594                 }
595                 if (opt->ts_needaddr) {
596                         optptr = raw + opt->ts;
597                         ip_rt_get_source(&optptr[optptr[2]-9], skb, rt);
598                         opt->is_changed = 1;
599                 }
600         }
601         if (opt->is_changed) {
602                 opt->is_changed = 0;
603                 ip_send_check(ip_hdr(skb));
604         }
605 }
606
607 int ip_options_rcv_srr(struct sk_buff *skb)
608 {
609         struct ip_options *opt = &(IPCB(skb)->opt);
610         int srrspace, srrptr;
611         __be32 nexthop;
612         struct iphdr *iph = ip_hdr(skb);
613         unsigned char *optptr = skb_network_header(skb) + opt->srr;
614         struct rtable *rt = skb_rtable(skb);
615         struct rtable *rt2;
616         unsigned long orefdst;
617         int err;
618
619         if (!rt)
620                 return 0;
621
622         if (skb->pkt_type != PACKET_HOST)
623                 return -EINVAL;
624         if (rt->rt_type == RTN_UNICAST) {
625                 if (!opt->is_strictroute)
626                         return 0;
627                 icmp_send(skb, ICMP_PARAMETERPROB, 0, htonl(16<<24));
628                 return -EINVAL;
629         }
630         if (rt->rt_type != RTN_LOCAL)
631                 return -EINVAL;
632
633         for (srrptr = optptr[2], srrspace = optptr[1]; srrptr <= srrspace; srrptr += 4) {
634                 if (srrptr + 3 > srrspace) {
635                         icmp_send(skb, ICMP_PARAMETERPROB, 0, htonl((opt->srr+2)<<24));
636                         return -EINVAL;
637                 }
638                 memcpy(&nexthop, &optptr[srrptr-1], 4);
639
640                 orefdst = skb->_skb_refdst;
641                 skb_dst_set(skb, NULL);
642                 err = ip_route_input(skb, nexthop, iph->saddr, iph->tos, skb->dev);
643                 rt2 = skb_rtable(skb);
644                 if (err || (rt2->rt_type != RTN_UNICAST && rt2->rt_type != RTN_LOCAL)) {
645                         skb_dst_drop(skb);
646                         skb->_skb_refdst = orefdst;
647                         return -EINVAL;
648                 }
649                 refdst_drop(orefdst);
650                 if (rt2->rt_type != RTN_LOCAL)
651                         break;
652                 /* Superfast 8) loopback forward */
653                 iph->daddr = nexthop;
654                 opt->is_changed = 1;
655         }
656         if (srrptr <= srrspace) {
657                 opt->srr_is_hit = 1;
658                 opt->nexthop = nexthop;
659                 opt->is_changed = 1;
660         }
```

```
661         return 0;
662 }
663 EXPORT_SYMBOL(ip_options_rcv_srr);
664
```