```
**********************************************************
   10565 Thu Jan  2 23:44:41 2014
new/usr/src/lib/libwrap/tcpd.h
4385 Missing hosts_ctl() prototype in tcpd.h
**********************************************************
```

```
/*
 * Copyright 2014 Sachidananda Urs <sacchi@gmail.com>
#endif /* ! codereview */
 * Copyright 2001 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 /*
 * @(#) tcpd.h 1.5 96/03/19 16:22:24
 *
 * Author: Wietse Venema, Eindhoven University of Technology, The Netherlands.
 */

#ifndef _TCPD_H
#define _TCPD_H

#endif /* ! codereview */
/*
 * HAVE_IPV6 is traditionally configured at tcp_wrappers build time but for
 * Solaris it must always be defined to keep the library interface binary
 * compatible.
 */
#define HAVE_IPV6

/* Structure to describe one communications endpoint. */

#define STRING_LENGTH   128             /* hosts, users, processes */

#include <sys/socket.h>
#include <netinet/in.h>

typedef struct sockaddr_gen {
    union {
        struct sockaddr _sg_sa;
        struct sockaddr_in      _sg_sin;
#ifdef HAVE_IPV6
        struct sockaddr_in6     _sg_sin6;
#endif
        } sg_addr;
} sockaddr_gen;

typedef union gen_addr {
    struct in_addr      ga_in;
#ifdef HAVE_IPV6
    struct in6_addr     ga_in6;
#endif
} gen_addr;

extern void sockgen_simplify();

#define sg_sa           sg_addr._sg_sa
#define sg_sin          sg_addr._sg_sin
#define sg_sin6         sg_addr._sg_sin6
#define sg_family       sg_sa.sa_family
#ifdef HAVE_IPV6
#define SGADDRSZ(sag)           ((sag)->sg_family == AF_INET6 ? \
                                    sizeof (struct in6_addr) : \
                                    sizeof (struct in_addr))
#define SGSOCKADDRSZ(sag)       ((sag)->sg_family == AF_INET6 ? \
```



```
                                    sizeof (struct sockaddr_in6) : \
                                    sizeof (struct sockaddr_in))
#define SGPORT(sag)             (*((sag)->sg_family == AF_INET6 ? \
                                    &(sag)->sg_sin6.sin6_port : \
                                    &(sag)->sg_sin.sin_port))
#define SGADDRP(sag)            (((sag)->sg_family == AF_INET6 ? \
                                    (char *)&(sag)->sg_sin6.sin6_addr : \
                                    (char *)&(sag)->sg_sin.sin_addr))
                                    (char *) &(sag)->sg_sin6.sin6_addr : \
                                    (char *) &(sag)->sg_sin.sin_addr))
#define SGFAM(sag)              ((sag)->sg_family == AF_INET6 ? \
                                    AF_INET6 : AF_INET)

#define SG_IS_UNSPECIFIED(sag) \
                ((sag)->sg_family == AF_INET6 ? \
                        IN6_IS_ADDR_UNSPECIFIED(&(sag)->sg_sin6.sin6_addr) : \
                        (sag)->sg_sin.sin_addr.s_addr == 0)

#define VALID_ADDRTYPE(t)       ((t) == AF_INET || (t) == AF_INET6)

#ifndef IPV6_ABITS
#define IPV6_ABITS 128                  /* Size of IPV6 address in bits */
#endif

#else /* HAVE_IPV6 */

#define SGADDRSZ(sag)           sizeof (struct in_addr)
#define SGSOCKADDRSZ(sag)       sizeof (struct sockaddr_in)
#define SGADDRSZ(sag)           sizeof(struct in_addr)
#define SGSOCKADDRSZ(sag)       sizeof(struct sockaddr_in)
#define SGPORT(sag)             ((sag)->sg_sin.sin_port)
#define SGADDRP(sag)            ((char *)&(sag)->sg_sin.sin_addr)
#define SGADDRP(sag)            ((char*) &(sag)->sg_sin.sin_addr)
#define SGFAM(sag)              AF_INET
#define SG_IS_UNSPECIFIED(sag)  ((sag)->sg_sin.sin_addr.s_addr == 0)

#define VALID_ADDRTYPE(t)       ((t) == AF_INET)

#endif /* HAVE_IPV6 */

struct host_info {
    char    name[STRING_LENGTH];        /* access via eval_hostname(host) */
    char    addr[STRING_LENGTH];        /* access via eval_hostaddr(host) */
    struct sockaddr_gen *sin;           /* socket address or 0 */
    struct t_unitdata *unit;            /* TLI transport address or 0 */
    struct request_info *request;       /* for shared information */
};
______unchanged_portion_omitted_

/* Common string operations. Less clutter should be more readable. */

#define STRN_CPY(d, s, l)       { strncpy((d), (s), (l)); (d)[(l)-1] = 0; }
#define STRN_CPY(d,s,l) { strncpy((d),(s),(l)); (d)[(l)-1] = 0; }

#define STRN_EQ(x, y, l)        (strncasecmp((x), (y), (l)) == 0)
#define STRN_NE(x, y, l)        (strncasecmp((x), (y), (l)) != 0)
#define STR_EQ(x, y)            (strcasecmp((x), (y)) == 0)
#define STR_NE(x, y)            (strcasecmp((x), (y)) != 0)
#define STRN_EQ(x,y,l)  (strncasecmp((x),(y),(l)) == 0)
#define STRN_NE(x,y,l)  (strncasecmp((x),(y),(l)) != 0)
#define STR_EQ(x,y)     (strcasecmp((x),(y)) == 0)
#define STR_NE(x,y)     (strcasecmp((x),(y)) != 0)

/*
 /*
 * Initially, all above strings have the empty value. Information that
```

```
 130  * cannot be determined at runtime is set to "unknown", so that we can
 131  * distinguish between 'unavailable' and 'not yet looked up'. A hostname
 132  * that we do not believe in is set to "paranoid".
 133  */

 135 #define STRING_UNKNOWN  "unknown"       /* lookup failed */
 136 #define STRING_PARANOID "paranoid"      /* hostname conflict */

 138 extern char unknown[];
 139 extern char paranoid[];

 141 #define HOSTNAME_KNOWN(s) (STR_NE((s), unknown) && STR_NE((s), paranoid))
  85 #define HOSTNAME_KNOWN(s) (STR_NE((s),unknown) && STR_NE((s),paranoid))

 143 #ifdef HAVE_IPV6
 144 #define NOT_INADDR(s) (strchr(s, ':') == 0 && s[strspn(s, "0123456789./")] != 0)
  88 #define NOT_INADDR(s) (strchr(s,':') == 0 && s[strspn(s,"0123456789./")] != 0)
 145 #else
 146 #define NOT_INADDR(s) (s[strspn(s, "0123456789./")] != 0)
  90 #define NOT_INADDR(s) (s[strspn(s,"0123456789./")] != 0)
 147 #endif

 149 /* Global functions. */

 151 #if defined(TLI) || defined(PTX) || defined(TLI_SEQUENT)
 152 extern void fromhost();                 /* get/validate client host info */
 153 #else
 154 #define fromhost sock_host              /* no TLI support needed */
 155 #endif

 157 extern int hosts_ctl();                 /* wrapper around request_init() */
 158 #endif /* ! codereview */
 159 extern int hosts_access();              /* access control */
 160 extern void shell_cmd();                /* execute shell command */
 161 extern char *percent_x();               /* do %<char> expansion */
 162 extern void rfc931();                   /* client name from RFC 931 daemon */
 163 extern void clean_exit();               /* clean up and exit */
 164 extern void refuse();                   /* clean up and exit */
 165 extern char *xgets();                   /* fgets() on steroids */
 166 extern char *split_at();                /* strchr() and split */
 167 extern unsigned long dot_quad_addr();   /* restricted inet_addr() */
 168 extern int numeric_addr();              /* IP4/IP6 inet_addr (restricted) */
 169 extern struct hostent *tcpd_gethostbyname();
 170                                         /* IP4/IP6 gethostbyname */
 171 #ifdef HAVE_IPV6
 172 extern char *skip_ipv6_addrs();         /* skip over colons in IPv6 addrs */
 173 #else
 174 #define skip_ipv6_addrs(x)      x
 175 #endif

 177 /* Global variables. */

 179 extern int allow_severity;              /* for connection logging */
 180 extern int deny_severity;               /* for connection logging */
 181 extern char *hosts_allow_table;         /* for verification mode redirection */
 182 extern char *hosts_deny_table;          /* for verification mode redirection */
 183 extern int hosts_access_verbose;        /* for verbose matching mode */
 184 extern int rfc931_timeout;              /* user lookup timeout */
 185 extern int resident;                    /* > 0 if resident process */

 187 /*
 101  /*
 188  * Routines for controlled initialization and update of request structure
 189  * attributes. Each attribute has its own key.
 190  */
```



```
 192 #ifdef __STDC__
 193 extern struct request_info *request_init(struct request_info *, ...);
 194 extern struct request_info *request_set(struct request_info *, ...);
 107 extern struct request_info *request_init(struct request_info *,...);
 108 extern struct request_info *request_set(struct request_info *,...);
 195 #else
 196 extern struct request_info *request_init();     /* initialize request */
 197 extern struct request_info *request_set();      /* update request structure */
 198 #endif

 200 #define RQ_FILE         1               /* file descriptor */
 201 #define RQ_DAEMON       2               /* server process (argv[0]) */
 202 #define RQ_USER         3               /* client user name */
 203 #define RQ_CLIENT_NAME  4               /* client host name */
 204 #define RQ_CLIENT_ADDR  5               /* client host address */
 205 #define RQ_CLIENT_SIN   6               /* client endpoint (internal) */
 206 #define RQ_SERVER_NAME  7               /* server host name */
 207 #define RQ_SERVER_ADDR  8               /* server host address */
 208 #define RQ_SERVER_SIN   9               /* server endpoint (internal) */

 210 /*
 124  /*
 211  * Routines for delayed evaluation of request attributes. Each attribute
 212  * type has its own access method. The trivial ones are implemented by
 213  * macros. The other ones are wrappers around the transport-specific host
 214  * name, address, and client user lookup methods. The request_info and
 215  * host_info structures serve as caches for the lookup results.
 216  */

 218 extern char *eval_user();               /* client user */
 219 extern char *eval_hostname();           /* printable hostname */
 220 extern char *eval_hostaddr();           /* printable host address */
 221 extern char *eval_hostinfo();           /* host name or address */
 222 extern char *eval_client();             /* whatever is available */
 223 extern char *eval_server();             /* whatever is available */
 224 #define eval_daemon(r)  ((r)->daemon)   /* daemon process name */
 225 #define eval_pid(r)     ((r)->pid)      /* process id */

 227 /* Socket-specific methods, including DNS hostname lookups. */

 229 extern void sock_host();                /* look up endpoint addresses */
 230 extern void sock_hostname();            /* translate address to hostname */
 231 extern void sock_hostaddr();            /* address to printable address */
 232 #define sock_methods(r) \
 233         { (r)->hostname = sock_hostname; (r)->hostaddr = sock_hostaddr; }

 235 /* The System V Transport-Level Interface (TLI) interface. */

 237 #if defined(TLI) || defined(PTX) || defined(TLI_SEQUENT)
 238 extern void tli_host();                 /* look up endpoint addresses etc. */
 239 #endif

 241 /*
 155  /*
 242  * Problem reporting interface. Additional file/line context is reported
 243  * when available. The jump buffer (tcpd_buf) is not declared here, or
 244  * everyone would have to include <setjmp.h>.
 245  */

 247 #ifdef __STDC__
 248 extern void tcpd_warn(char *, ...);     /* report problem and proceed */
 249 extern void tcpd_jump(char *, ...);     /* report problem and jump */
 250 #else
 251 extern void tcpd_warn();
 252 extern void tcpd_jump();
 253 #endif
```

```
255 struct tcpd_context {
256     char   *file;                   /* current file */
257     int     line;                   /* current line */
258 };
259 extern struct tcpd_context tcpd_context;

261 /*
175  /*
262  * While processing access control rules, error conditions are handled by
263  * jumping back into the hosts_access() routine. This is cleaner than
264  * checking the return value of each and every silly little function. The
265  * (-1) returns are here because zero is already taken by longjmp().
266  */

268 #define AC_PERMIT       1               /* permit access */
269 #define AC_DENY         (-1)            /* deny_access */
270 #define AC_ERROR        AC_DENY         /* XXX */

272 /*
186  /*
273  * In verification mode an option function should just say what it would do,
274  * instead of really doing it. An option function that would not return
275  * should clear the dry_run flag to inform the caller of this unusual
276  * behavior.
277  */

279 extern void process_options();          /* execute options */
280 extern int dry_run;                     /* verification flag */

282 /* Bug workarounds. */

284 #ifdef INET_ADDR_BUG                    /* inet_addr() returns struct */
285 #define inet_addr fix_inet_addr
286 extern long fix_inet_addr();
287 #endif

289 #ifdef BROKEN_FGETS                     /* partial reads from sockets */
290 #define fgets fix_fgets
291 extern char *fix_fgets();
292 #endif

294 #ifdef RECVFROM_BUG                     /* no address family info */
295 #define recvfrom fix_recvfrom
296 extern int fix_recvfrom();
297 #endif

299 #ifdef GETPEERNAME_BUG                  /* claims success with UDP */
300 #define getpeername fix_getpeername
301 extern int fix_getpeername();
302 #endif

304 #ifdef SOLARIS_24_GETHOSTBYNAME_BUG     /* lists addresses as aliases */
305 #define gethostbyname fix_gethostbyname
306 extern struct hostent *fix_gethostbyname();
307 #endif

309 #ifdef USE_STRSEP                       /* libc calls strtok() */
310 #define strtok  fix_strtok
311 extern char *fix_strtok();
312 #endif

314 #ifdef LIBC_CALLS_STRTOK                /* libc calls strtok() */
315 #define strtok  my_strtok
316 extern char *my_strtok();
317 #endif
```



```
319 #endif /* _TCPD_H */
320 #endif /* ! codereview */
```