

eatables\_u.h

<pre>/*  * \$Id: etables.c,v 1.03 2002/01/19  * Copyright (C) 2001-2002 Bart De Schuymer  *  * This code is stongly inspired on the iptables code which  * is  * Copyright (C) 1999 Paul 'Rusty' Russell &amp; Michael J. Neu  * ling  *  * This program is free software; you can redistribute it an  * d/or  * modify it under the terms of the GNU General Public Licen  * se as  * published by the Free Software Foundation; either version  * 2 of the  * License, or (at your option) any later version.  *  * This program is distributed in the hope that it will be u  * seful, but  * WITHOUT ANY WARRANTY; without even the implied warranty o  * f  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See  * the GNU  * General Public License for more details.  *  * You should have received a copy of the GNU General Public  * License  * along with this program; if not, write to the Free Softwa  * re  * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.  */  #ifndef ETABLES_U_H #define ETABLES_U_H #include &lt;netinet/in.h&gt; #include &lt;linux/netfilter_bridge/etables.h&gt;  struct ebt_u_entries {     int policy;     unsigned int nentries;     /* counter offset for this chain */     unsigned int counter_offset;     /* used for udc */     unsigned int hook_mask;     char name[EBT_CHAIN_MAXNAMELEN];     struct ebt_u_entry *entries; };  struct ebt_u_chain_list {     struct ebt_u_entries *udc;     struct ebt_u_chain_list *next;     /* this is only used internally, in communication.c */ };  char *kernel_start;  struct ebt_u_replace {     char name[EBT_TABLE_MAXNAMELEN];     unsigned int valid_hooks;</pre>	<pre>/* nr of rules in the table */ unsigned int nentries; struct ebt_u_entries *hook_entry[NF_BR_NUMHOOKS]; /* user defined chains (udc) list */ struct ebt_u_chain_list *udc; /* nr of counters userspace expects back */ unsigned int num_counters; /* where the kernel will put the old counters */ struct ebt_counter *counters; /*  * can be used e.g. to know if a standard option  * has been specified twice  */ unsigned int flags; /* we stick the specified command (e.g. -A) in here char command; /*  * here we stick the hook to do our thing on (can be -1 if unspecified)  */ int selected_hook; /* used for the atomic option */ char *filename; /* tells what happened to the old rules */ unsigned short *counterchanges; };  struct ebt_u_table {     char name[EBT_TABLE_MAXNAMELEN];     void (*check)(struct ebt_u_replace *repl);     void (*help)(char **);     struct ebt_u_table *next; };  struct ebt_u_match_list {     struct ebt_u_match_list *next;     struct ebt_entry_match *m; };  struct ebt_u_watcher_list {     struct ebt_u_watcher_list *next;     struct ebt_entry_watcher *w; };  struct ebt_u_entry {     unsigned int bitmask;     unsigned int invflags;     uint16_t ethproto;     char in[IFNAMSIZ];     char logical_in[IFNAMSIZ];     char out[IFNAMSIZ];     char logical_out[IFNAMSIZ];     unsigned char source_mac[ETH_ALEN];     unsigned char source_mac[ETH_ALEN];     unsigned char dest_mac[ETH_ALEN];     unsigned char dest_mask[ETH_ALEN];     struct ebt_u_match_list *m_list;     struct ebt_u_watcher_list *w_list;</pre>	<pre>    struct ebt_entry_target *t;     struct ebt_u_entry *next; };  struct ebt_u_match {     char name[EBT_FUNCTION_MAXNAMELEN];     /* size of the real match data */     unsigned int size;     void (*help)(void);     void (*init)(struct ebt_entry_match *m);     int (*parse)(int c, char **argv, int argc,         const struct ebt_u_entry *entry, unsigned in t *flags,         struct ebt_entry_match **match);     void (*final_check)(const struct ebt_u_entry *entry,         const struct ebt_entry_match *match,         const char *name, unsigned int hookmask, unsigned int time);     void (*print)(const struct ebt_u_entry *entry,         const struct ebt_entry_match *match);     int (*compare)(const struct ebt_entry_match *m1,         const struct ebt_entry_match *m2);     const struct option *extra_ops;     /*      * can be used e.g. to check for multiple occurrence      of the same option      */     unsigned int flags;     unsigned int option_offset;     struct ebt_entry_match *m;     /*      * if used == 1 we no longer have to add it to      * the match chain of the new entry      */     unsigned int used;     struct ebt_u_match *next; };  struct ebt_u_watcher {     char name[EBT_FUNCTION_MAXNAMELEN];     unsigned int size;     void (*help)(void);     void (*init)(struct ebt_entry_watcher *w);     int (*parse)(int c, char **argv, int argc,         const struct ebt_u_entry *entry, unsigned int *fl ags,         struct ebt_entry_watcher **watcher);     void (*final_check)(const struct ebt_u_entry *entry,         const struct ebt_entry_watcher *watch, const char         *name,         unsigned int hookmask, unsigned int time);     void (*print)(const struct ebt_u_entry *entry,         const struct ebt_entry_watcher *watcher);     int (*compare)(const struct ebt_entry_watcher *w1,         const struct ebt_entry_watcher *w2);     const struct option *extra_ops;     unsigned int flags;     unsigned int option_offset;     struct ebt_entry_watcher *w;     unsigned int used;     struct ebt_u_watcher *next;</pre>
---	---	--

## ebtables\_u.h

```

};

struct ebt_u_target
{
    char name[EBT_FUNCTION_MAXNAMELEN];
    unsigned int size;
    void (*help)(void);
    void (*init)(struct ebt_entry_target *t);
    int (*parse)(int c, char **argv, int argc,
        const struct ebt_u_entry *entry, unsigned int *fl

ags,
        struct ebt_entry_target **target);
    void (*final_check)(const struct ebt_u_entry *entry,
        const struct ebt_entry_target *target, const char

*name,
        unsigned int hookmask, unsigned int time);
    void (*print)(const struct ebt_u_entry *entry,
        const struct ebt_entry_target *target);
    int (*compare)(const struct ebt_entry_target *t1,
        const struct ebt_entry_target *t2);
    const struct option *extra_ops;
    unsigned int option_offset;
    unsigned int flags;
    struct ebt_entry_target *t;
    unsigned int used;
    struct ebt_u_target *next;
};

void register_table(struct ebt_u_table *);
void register_match(struct ebt_u_match *);
void register_watcher(struct ebt_u_watcher *);
void register_target(struct ebt_u_target *t);
int get_table(struct ebt_u_replace *repl);
struct ebt_u_target *find_target(const char *name);
struct ebt_u_match *find_match(const char *name);
struct ebt_u_watcher *find_watcher(const char *name);
struct ebt_u_table *find_table(char *name);
void deliver_counters(struct ebt_u_replace *repl);
void deliver_table(struct ebt_u_replace *repl);
void check_option(unsigned int *flags, unsigned int mask);
int check_inverse(const char option[]);
void print_mac(const char *mac);
void print_mac_and_mask(const char *mac, const char *mask);
int ebtables_insmod(const char *modname);
void __print_bug(char *file, int line, char *format, ...);
#define print_bug(format, args...) \
    __print_bug(__FILE__, __LINE__, format, ##args)
#define print_error(format, args...) {printf(format, ##args);\
    printf("\n");exit(-1);}
#define print_memory() {printf("Ebtables: " __FILE__ \
    " %s %d :Out of memory.\n", __FUNCTION__, __LINE__); exit
    (-1);}

/* used for keeping the rule counters right during rule adds
or deletes */
#define CNT_NORM 0
#define CNT_DEL 1
#define CNT_ADD 2
#define CNT_END 3
#define CNT_ZERO 4

extern char *standard_targets[NUM_STANDARD_TARGETS];
/*

```

```

* Transforms a target string into the right integer,
* returns 0 on success.
*/
#define FILL_TARGET(_str, _pos) ({
    int _i, _ret = 0;
    for (_i = 0; _i < NUM_STANDARD_TARGETS; _i++)
        if (!strcmp(_str, standard_targets[_i])) {
            _pos = _i - 1;
            break;
        }
    if (_i == NUM_STANDARD_TARGETS)
        _ret = 1;
    _ret;
})

/* Transforms the target value to an index into standard_tar
gets[] */
#define TARGET_INDEX(_value) (-_value - 1)
/* Returns a target string corresponding to the value */
#define TARGET_NAME(_value) (standard_targets[TARGET_INDEX(_
value)])
/* True if the hook mask denotes that the rule is in a base
chain */
#define BASE_CHAIN (hookmask & (1 << NF_BR_NUMHOOKS))
/* Clear the bit in the hook_mask that tells if the rule is
on a base chain */
#define CLEAR_BASE_CHAIN_BIT (hookmask &= ~(1 << NF_BR_NUMHO
OKS))
#endif /* EBTABLES_U_H */

```